On the Construction, Dimensionality, and Decoding of Linear Block Code Trellises

by
Alan D. Kot
B.A.Sc. (Hons.), The University of British Columbia, 1981
M.A.Sc., The University of British Columbia, 1987

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF DOCTOR OF PHILOSOPHY
in
THE FACULTY OF GRADUATE STUDIES
(Department of Electrical Engineering)

We accept this thesis as conforming

to the required standard

THE UNIVERSITY OF BRITISH COLUMBIA

December 1992

In presenting this thesis in partial fulfilment of the requirements for an advanced degree at the University of British Columbia, I agree that the Library shall make it freely available for reference and study. I further agree that permission for extensive copying of this thesis for scholarly purposes may be granted by the head of my department or by his or her representatives. It is understood that copying or publication of this thesis for financial gain shall not be allowed without my written permission.

(Signature) _____

Department of Electrical Engineering

The University of British Columbia
Vancouver, Canada

Date Apr. 16, 1993

Abstract

In principle, the coding gain of a digital communications system can be improved through the use of longer error control codes and soft-decision maximum-likelihood (ML) decoding. Unfortunately, the implementation of such techniques is limited by the computational requirements of the decoder. In this thesis, we consider several aspects of trellis-based ML, and 'near-ML', soft-decision decoding of general linear block codes.

ML decoding is feasible for reasonably compact trellises using the Viterbi algorithm. For general linear block codes we present simple methods for trellis construction that are based on the methods of Wolf and Massey. It is confirmed that the trellises so constructed are minimal, and an improvement of Muder's lower bound on the maximum trellis dimension is found. It is shown that when equivalent codes are constructed by permutations of the symbol positions that the resulting trellis dimensions are fixed near either end, while in the central portion of the trellis the dimensions vary between Wolf's upper bound on the maximum dimension and a new lower bound on the minimum dimension. This lower bound and the improved bound on the maximum trellis dimension are exact for maximum distance separable codes. These bounds indicate that only codes (and their duals) that have a smallest minimum distance $\min\left(d_{min}, d_{min}^{\perp}\right)$ significantly less than the corresponding Singleton bound can possibly have a compact trellis.

For trellises that are impractically large for decoding by the Viterbi algorithm, we consider near-ML decoding using partial searches of a code trellis or tree. Despite the fact that this approach is suboptimal in terms of the probability of error, it has the potential for an improved trade-off between coding gain and computational effort. The decoding problem faced by a partial trellis search is described in an alternative manner to Massey's variable-length code model. As part of this approach, we specialize Massey's use of 'random-tails' in order to exploit *a priori* knowledge of the code and information-symbol distribution used. As a result there are some minor changes to the usual

metric, but only in atypical situations — such as the use of unequal *a priori* information-symbol probabilities, non-linear codes, and non-symmetric, non-binary discrete memoryless channels.

We introduce a simple and efficient (linear time) method for the selection of the best metrics from among a number of contenders in a breadth-first search. This method can provide a sorted set of survivor metrics during M algorithm decoding of binary linear block codes using only M comparisons in the worst case. The application of the method to the decoding of convolutional codes is also discussed.

We present a method for soft-decision decoding of general linear block codes that we refer to as *Reconfigurable Trellis (RT) Decoding*. RT decoding carries out a partial search on a different and more easily searched trellis (or tree) for the code. The trellis used for decoding is dependent upon the reliability of the received data, so that it is determined 'on-the-fly'. Only a portion of the reconfigured trellis needs to be constructed, as guided by the partial search. While RT decoding requires some computational overhead prior to beginning each search, the search effort itself can be very significantly reduced, so that overall an improvement can be achieved in the trade-off between coding gain and computational effort.

With respect to the performance analysis of suboptimal soft-decision decoding schemes, we extend the uniform error property of ML decoding of linear codes on a binary-input symmetric-output memoryless channel to include the more general case of partial trellis searches. For RT-M decoding of general linear block codes the uniform error property is used together with a novel channel model and a very simple search technique to estimate the increase in the probability of error over that of ML decoding. This approach can yield accurate results while avoiding the complexity of considering numerous soft-decision output values, and without requiring detailed knowledge of the code structure.

Table of Contents

List of Figures

List of Tables

Acknowledgements

I wish to sincerely thank my research supervisor, Professor Cyril Leung, for his continual interest and highly constructive criticism.

I would also like to thank the following people who kindly provided me with preprints of their work; J.B. Anderson, J.T. Coffey and R.M. Goodman, F. Hemmati, and J.L Massey.

Finally, I am grateful for financial support received from a Natural Sciences and Engineering Research Council (NSERC) Postgraduate Scholarship, a British Columbia Science Council G.R.E.A.T. Scholarship, a British Columbia Advanced Systems Institute Scholarship, a University of British Columbia Supplemental Scholarship, and from NSERC Grant OGP–1731.

Chapter 1

Introduction

Never discard information prematurely...

A.J. Viterbi [1]

THE reliability and efficiency of digital communications systems can be significantly improved through the use of error control coding techniques. Although error control coding is one of several potential techniques — including various modulation, diversity (time, space, frequency, etc.), and retransmission schemes — these techniques are complementary. Communications systems can employ some combination of these techniques to trade off reliability against various costs such as bandwidth, delay, transmitted power, and hardware requirements.

In implementations of error control coding schemes, the complexity of the decoder is typically the limiting constraint. However, improvements in decoding algorithms, in concert with the increasing performance and decreasing costs of VLSI circuit technology, facilitate the utilization of more powerful error control codes. The work presented in this thesis focuses on the decoding

'bottleneck', and we consider efficient approaches to attain *optimal* and *near-optimal* decoding of general linear block codes. By 'optimal', we mean that a decoder's probability of error is that of a maximum-likelihood (ML) decoder. The motivation in considering 'near-optimal' decoders is to obtain a significant decrease in decoding effort while incurring a negligible or a tolerable increase in probability of error. As well, the motivation for considering *general* linear block codes is to have a single method accommodate several applications, so that improved economies-of-scale may be realized.

1.1 The Decoding Problem

Since we are interested in general codes, it is useful to consider the results of information theory in guiding us towards useful codes. We recall the fact that improved performance is generally obtained for longer codes. This is a direct result of the channel coding theorem, which in part states that the expected probability of message error is upper bounded by [2]

$$\overline{P_e} \leq e^{-nE_r(R)}, \tag{1.1}$$

where n is the block length, R is the code rate, and $E_r(R)$ is the random coding exponent. This result applies to the ensemble of all randomly selected codes, thus guaranteeing that codes exist that satisfy the bound of (1.1). For rates R below the capacity of the channel, $E_r(R) > 0$, and thus a longer block length will improve ensemble code performance.

Unfortunately, the desirable error control power of the longer codes comes mainly at a cost of greater code complexity. This has constrained implementations to moderate blocklengths. The objective is then not only to develop efficient decoding algorithms but to explore the trade-off between the decoding effort and the probability of error. Different decoding algorithms can have

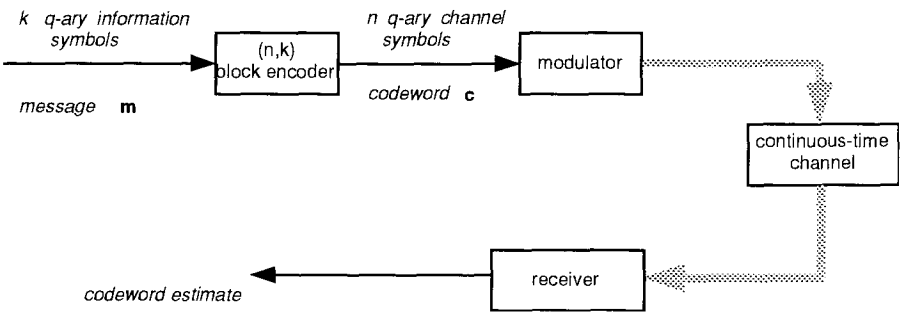


Figure 1.1. Simplified Diagram of a Block Coded Communication System

substantial differences in error probability and in decoding effort when decoding the same code. If a desired error probability is specified one should seek the *combination* of code and decoder that minimize the decoding effort. In the absence of a performance specification, one should seek decoders that are *good* in the sense of attaining low error probability with low decoding effort, when operating on a specific code.

A simplified diagram of a communication system that employs error control coding is shown in Figure 1.1. The (n, k) block encoder accepts k q-ary information symbols at a time, and forms a codeword $\mathbf{c} = (\, c_1 \quad c_2 \quad \cdots \quad c_n \,)$ which is fed to the modulator. The modulator maps each codeword into a continuous-time waveform, which is transmitted through the channel where it is subject to degradation. The task of the receiver is to estimate from the channel output which codeword was transmitted. We take as the criterion of optimality the probability of codeword error. Ideally a *maximum a posteriori* (MAP) estimate of the transmitted codeword is formed, as this minimizes the codeword error probability [3]. A brute force method of achieving this is to compute an *a posteriori* probability for each codeword, and then select the largest. Since there are q^k codewords this brute force method is usually impractical. To decode more efficiently, a closer examination of the problem is required.

1.2 Hard-Decisions, Soft-Decisions, and the Discarding of Likelihood Information

Figure 1.2 shows a system similar to Figure 1.1, except that four types of decoding schemes are shown. For practical decoding the receiver processing is split into several sub-blocks, as shown, with most of the blocks operating on discrete-time data.[1] The purpose of Figure 1.2 is to display the salient differences between the various decoders in terms of where information is discarded, either by quantization or by intentional omission. In Figure 1.2, blocks that quantize or otherwise discard information are shown shaded.

The demodulator delivers vectors of data to be processed by the decoder. The n vectors of demodulator output $\mathbf{r}_i$ are denoted as $\mathbf{r} = (\,\mathbf{r}_1 \quad \mathbf{r}_2 \quad \cdots \quad \mathbf{r}_n\,)$. The demodulator output may include channel state information during each symbol interval, which can improve performance.[2]

Consider the vector of demodulator output $\mathbf{r}_i$ for the i^{th} symbol. Let the demodulator output be processed to yield symbol likelihoods, which can always be done without losing any relevant information [5]. This maps a $\mathbf{r}_i$ into q symbol likelihoods $(\ f(\mathbf{r}_i \mid 0),\ f(\mathbf{r}_i \mid 1), \ldots,\ f(\mathbf{r}_i \mid q-1)\ )$. Each of the four decoder strategies shown in Figure 1.2 is shown to have a likelihood computation block which computes, for each symbol period, q symbol likelihoods. These q likelihoods per symbol period are ideally all passed to the next stage of the decoder. Decoder strategies differ significantly in the manner in which they handle these likelihoods, as discussed below.

A common simplification that is made to MAP decoding is to assume equally likely codewords, in which case the MAP decoding rule simplifies to a Maximum Likelihood (ML) decoding rule

[1] In practice, some of the blocks may not be explicitly separate as shown here (e.g. the likelihood computation block) since their function may be handled implicitly by the demodulator or decoder.

[2] A common example of channel state information is the estimation of the received signal level on a Rayleigh fading channel. For channel state information we assume that the channel and modulation are of the SOCFES (State Observable Channel with Freely Evolving State) type, described in [4]. The requirements for a SOCFES are that the observable channel states be independent of the modulated symbols and that the probability distribution of the demodulator output for a symbol period depends only on the hypothesized transmitted symbol and the observed channel state.

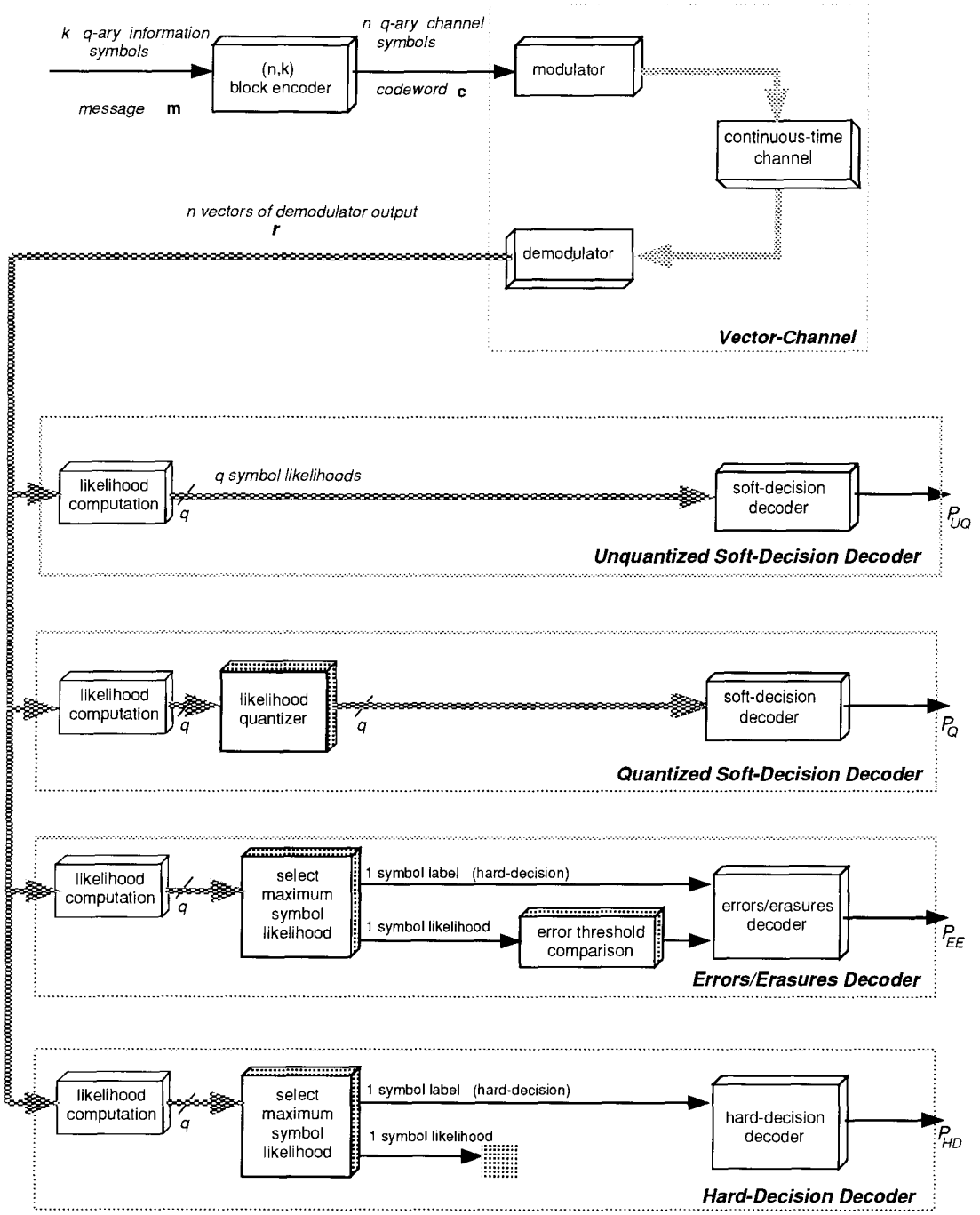


Figure 1.2. Some Block Decoder Strategies

Blocks with inherent performance losses due to quantization or ignoring some data are shown shaded. Dark shaded lines indicate vector(s) of data transferred per symbol period. The block error probabilities for the unquantized, errors/erasures, and hard-decision decoders satisfy $P_{UQ} \leq P_{EE} \leq P_{HD}$. As well, for typical quantized soft-decision decoders we have $P_{UQ} \leq P_Q \leq P_{EE} \leq P_{HD}$.

[3]. Under the assumption that the channel is memoryless[3], the ML decoding rule chooses the codeword $\mathbf{c}$ that maximizes

$$f(\mathbf{r}|\mathbf{c}) = \prod_{i=1}^{n} f(\mathbf{r}_i|\, c_i) \tag{1.2}$$

where each term in the product is the likelihood of the channel output $\mathbf{r}_i$ given the hypothesized symbol $\mathbf{c}_i$.

The unquantized soft-decision decoder shown in Figure 1.2 processes all of the exact (unquantized) symbol likelihoods so that it can potentially deliver a ML decision. Let P_{UQ} denote the probability that the decoder output word is in error.

The quantized soft-decision decoder processes symbol likelihoods after quantization, a process that has an inherently degrading effect on performance. While the effects of quantization can be made negligible, in a practical implementation where the goal is to minimize the number of quantization levels there will be some budgeted loss in performance, so that $P_Q \geq P_{UQ}$. If we view the likelihood quantizer block in Figure 1.2 as being rather versatile — in the sense that in addition to quantization it can perform scaling (i.e. multiplication by a constant), selection of a maximum, and make comparisons against a threshold — then we can view errors/erasures and hard-decision decoding as special cases of the quantized soft-decision decoder. The errors/erasures strategy is formed by having the quantizer select, per each symbol period, the symbol element that has the largest likelihood if the latter is above a certain threshold; otherwise all q symbol likelihoods are set to zero. This action can be separated into first selecting the maximum symbol likelihood, which is then followed by thresholding to 'flag' erasures, as shown in Figure 1.2. In this way we can see that information is discarded at both steps. The probability of error for the errors/erasures decoder is denoted as P_{EE}. Typically the errors/erasures strategy discards more

[3] Or, that the channel can be made memoryless by incorporating channel state information (see previous footnote).

likelihood information than the quantized soft-decision decoder, so that P_{EE} is typically greater than P_Q. (Note however that in cases where one uses a very poor quantizer, the errors/erasures decoder can outperform the quantized soft-decision decoder.) Finally, the hard-decision strategy is formed by having the quantizer select, per each symbol period, the symbol that has the largest likelihood and normalizing this likelihood to some standard value, then setting all other symbol element likelihoods to zero. This can be viewed as tossing out the erasure information of the errors/erasures decoder, as shown in Figure 1.2, so that $P_{HD} \geq P_{EE}$.

Comparing the block error probability of the four decoding strategies shown in Figure 1.2 we have $P_{UQ} \leq P_{EE} \leq P_{HD}$, and typically $P_{UQ} \leq P_Q \leq P_{EE} \leq P_{HD}$. The decrease in performance is concomitant with less likelihood information being passed to the final decoding block. It is hoped that the decrease in performance arising from discarding likelihood information prior to the final decoding block will be offset if there are significant savings in computational effort. The approach taken in this thesis, however, is to attempt to achieve substantial computational savings without discarding significant likelihood information prior to the final decoding block. We take the approach that *the soft-decision decoder itself should be the judge of when to discard likelihood information.* The motivation for this is twofold. First, retaining all of the likelihood information for use by the soft-decision decoder can avoid an irrecoverable loss of coding gain. Second, the soft-decision decoder may be able to exploit the soft-decisions to simplify its decoding task.

The loss of coding gain incurred by hard-decision decoding on the additive white Gaussian noise (AWGN) channel with binary antipodal signaling is approximately 2 dB, for the ensemble of randomly constructed block codes[4] [3]. If we consider specific codes instead of randomly constructed codes there are some other results available. For a code with guaranteed hard-decision

[4] As the SNR is decreased, the loss can be shown to approach $10\log_{10}(2/\pi) = 1.96$ dB.

error correcting capability of

$$t = \left\lfloor \frac{d_{min} - 1}{2} \right\rfloor, \tag{1.3}$$

Chase [6] has shown that as the SNR is increased, the use of ML soft-decision decoding will effectively increase the error correcting capability to $d_{min} - 1$. This corresponds to an approximate doubling of the error correcting power of the code given by (1.3). In terms of dB gain, the asymptotic improvement is 3 dB. At practical signal to noise ratios the gain obtained is closer to 2 dB.

More significant gains can be realized using soft-decision decoding on some other channels. Of considerable importance is an AWGN channel with Rayleigh faded signal amplitude and uniformly distributed phase. Such a channel arises in mobile radio propagation at VHF/UHF frequencies in urban areas [7]. Rayleigh amplitude fading was also considered by Chase, with binary antipodal signaling. It was shown that the gain can be significantly larger than 3 dB, and that the gain will increase further with SNR, since ML decoding results in a change in asymptotic behaviour from a $(t+1)^{\text{th}}$ order to a d_{min}^{th} order diversity system. Simulation results [6] for a (24,12) Golay code show a gain of approximately 6 dB relative to hard-decision decoding, at a bit error rate (BER) of 10^{-4}.

While we have reviewed that the fundamental motivation for using soft-decision decoding is that it can potentially exploit all of the available likelihood information, and in doing so attain the theoretical minimum probability of error, we have not yet suggested how this might be accomplished. The approach taken in this thesis is to employ tree and trellis representations of general linear block codes, which enable the decoder to easily utilize soft-decision metrics. Soft-decision decoding of convolutional codes has long exploited trees and trellises. ML decoding of linear block codes has also exploited trellises (e.g. [8]) using the Viterbi algorithm (VA), provided the trellis is reasonably compact. It is therefore of interest to know how to compute (and/or bound)

the trellis dimensions. As well, if the trellis is so large that a full search using the VA is impractical, a partial tree or trellis search could be considered. In this thesis we consider several aspects of general linear block code trellises, relating to their construction, dimensionality, and decoding via both complete and partial searches.

1.3 Organization of the Thesis

In Chapter 2 we discuss trellis representations of general linear block codes. Improved methods for constructing complete or partial trellises are presented, and new bounds on the trellis dimensions are found.

In Chapter 3 we consider the case where a full trellis search is impractical and investigate some aspects of partial trellis searches. We find an appropriate metric for discarding paths. We also present a new method to efficiently sort path metrics.

In Chapter 4 we present a class of decoding techniques that exploit soft-decisions — not only for the inherent coding gain — but to reduce the decoding effort. In doing so the techniques completely restructure the tree or trellis search.

In Chapter 5 we conclude with a summary of contributions made and suggest some topics for further research.

Chapter 2

Trellises for Linear Block Codes

*"The trellis is the **foundation** for channel coding in the sense that the trellis associated with the code is the primary determiner of the necessary effort for ML decoding."*
J.L. Massey [9].

TRELLIS representations of convolutional codes appear extensively in the literature. Originally, Forney pointed out that a convolutional code state diagram can be represented by a "trellis" [10][11]. With such a representation, ML decoding could then be viewed as a shortest-path problem, and it became clear that the Viterbi algorithm (VA) is optimal — not only asymptotically optimal as discussed in [12].[5] The powerful and versatile combination of the VA with convolutional code type trellises has since been successfully applied in many communications applications, including maximum-likelihood sequence estimation in intersymbol interference and the decoding of combined coding and modulation schemes [11][14].

[5] The optimality of the VA was also shown by Omura in [13].

In contrast, trellis representations of block codes have been discussed much less frequently, with only a few papers appearing within twenty years of the introduction of the convolutional code trellis. Of these, the introduction of linear block code trellises appears in [15][8][9]. Recently, in [16] and [17], trellis construction and the trellis state-space dimensions were re-examined. In this chapter, we continue this examination of trellis construction and dimensionality for general linear block codes.

In Section 2.1 we review some basic definitions related to trellises and their dimensions.

In Section 2.2 we review three methods for trellis construction and dimensional computations introduced by Wolf [8], Massey [9], and Forney [16]. It is shown that Massey's and Wolf's trellises are isomorphic[6] and the isomorphism is derived.

In Section 2.3 we discuss *minimal* trellises (trellises with a minimal number of states) and we utilize some of Muder's analysis in [17] to show that trellises generated using Wolf's or Massey's method yield minimal trellises. This complements Muder's result that Forney's method yields a minimal trellis [17].

In Section 2.4 three simplified trellis construction methods are presented for constructing minimal trellises of general linear block codes. Also, a method for computing the trellis dimensions is derived that is an alternative to the methods of [9] and [16].

In Section 2.5, we improve on a bound on the trellis dimensions due to Muder [17] and discuss some other properties of the trellis dimensions. In particular, it is shown that when equivalent codes are constructed by permutations of the symbol positions the resulting trellis dimensions are

---

[6] A trellis is *isomorphic* to some other trellis if it differs only in the labeling of states at each depth [17].

fixed near either end, while in the central portion of the trellis the dimensions vary between an attainable upper bound and a lower bound.

2.1 Code Trees and Trellises

In the communications literature, a *tree* or a *trellis* is commonly described as a graph that represents each codeword of a code by a distinct path, with all paths originating at a single *root* node. Each path is composed of *branches*, and a trellis differs from a tree in that its branches merge as the trellis is traversed. Each branch may be assigned a number of codeword symbols. For convolutional codes each branch usually represents the n channel symbols output during each shift of k information bits into a rate $R = k/n$ convolutional encoder. For block codes we usually associate one channel symbol per branch although this need not be the case [9][16][17]. Punctured convolutional [18][19] or block [20] codes are other examples where the resultant tree or trellis may have a non-constant number of channel symbols per branch.

We now describe trellises more formally. Most of this discussion is a summary of some definitions from Muder [17], with some alteration of notation.

Consider an edge labeled directed graph $T = (V, A, E)$ consisting of a set of vertices V and edges E, with each edge labeled by elements from an alphabet A. If V can be partitioned into a union of disjoint subsets

$$V = V_0 \cup V_1 \cup \cdots \cup V_n, \tag{2.1}$$

such that every edge that begins at a vertex in V_l ends at a vertex in V_{l+1}, then T is called a trellis of length n. The vertices V_l of a trellis are commonly called the states (or nodes) at depth (or level) l, and the edges are commonly called branches.

A codeword $\mathbf{c} \in \mathbf{C}$ is usually represented in T by a path consisting of n branches labeled by $c_1, c_2, \ldots, c_n$ from V_0 to V_n. As mentioned earlier, one may group branches to form multi-symbol branches.

An example of a trellis, under Muder's definition, is the usual code tree for C. We will denote the code tree for C as $\hat{T}$. The tree $\hat{T}$ is an example of a *proper* trellis, where a trellis is proper if V_0 has a single state **0** (the root), every state belongs to some length n path, and no two branches have the same initial state and label. For linear codes it is sufficient to limit our discussion to proper trellises [17].

The set of states at depth l in $\hat{T}$ correspond to the set of *heads* of length l. We introduce the term *head* to refer to a length l initial portion of a codeword, i.e. if $\mathbf{c} = (c_1, c_2, \ldots, c_l, c_{l+1} \ldots, c_n) \in \mathbf{C}$ then a head is $\mathbf{c}^h = (c_1, c_2, \ldots, c_l)$. We use Muder's terminology of referring to the *tail* as the length $n - l$ final portion of a codeword, denoted $\mathbf{c}^t$, so that when given a head $\mathbf{c}^h$ if

$$\left(\mathbf{c}^h, \mathbf{c}^t\right) \in \mathbf{C} \tag{2.2}$$

then $\mathbf{c}^t$ is called a tail of $\mathbf{c}^h$.

The number of states at each depth of the trellis is of interest, since this largely determines the ML decoding effort of both convolutional and block codes [9]. As in [17], let S_l denote the number of states at depth l of a trellis of C, and let $S = \max_l S_l$, $s_l = \log_q S_l$, and $s = \log_q S$. We refer to s_l as the dimension of the trellis at depth l, and s simply as the maximum trellis dimension.[7]

---

[7] We note that in [17] that these terms refer to *minimal trellises*, which will be discussed in §2.3. Our present use of these terms to refer to general trellises will not cause confusion, since at present we need not distinguish between minimal and non-minimal trellises. Moreover, it will become apparent in §2.3 and §2.4 that all trellises considered in this thesis are in fact minimal.

2.2 Review of Trellis Construction Methods

Trellis representations of convolutional codes arise naturally from the fact that the convolutional encoder is a finite state machine. To construct a convolutional code trellis one can assign trellis states to be a vector corresponding to the contents of the encoder shift register [10][11], and the branches are labeled with the corresponding encoded symbols.

Constructing trellises for general linear block codes is less obvious than for the case of convolutional codes. In this section we review trellis constructions for general linear block codes due to Wolf [8], Massey [9], and Forney [16], with emphasis on the first two methods since they form the basis for the simplified trellis construction methods discussed in §2.4.

2.2.1 Wolf's Trellis Construction Method

Trellis representations of linear block codes appear to have been first introduced by Bahl *et al* [15], who showed that linear block codes may be represented by a trellis and examined a decoding algorithm that minimizes the symbol error rate. Wolf [8] later discussed trellis construction in more detail and emphasized that ML block decoding can be achieved using the Viterbi algorithm.

Wolf's trellis construction method for a general linear block code is based on the parity check matrix.[8] We show how the trellis for a code can be generated by first describing how any particular codeword is represented. Let $\mathbf{H}$ denote the code's parity check matrix and let $\mathbf{h}_i$ denote the i^{th} column vector of $\mathbf{H}$. The generator matrix $\mathbf{G}$ corresponding to $\mathbf{H}$ contains rows that span a k dimensional subspace of GF$(q)^n$, and this subspace corresponds to the code. Recall that the row space of $\mathbf{H}$ is an orthogonal complement to the row space of $\mathbf{G}$, so that

$$\mathbf{H}\mathbf{G}^{\mathrm{T}} = \mathbf{0}. \tag{2.3}$$

[8] Wolf also discussed that a trellis for a cyclic block code can be generated in a manner similar to that for a convolutional code, in that one can take as the state the contents of the encoder shift register.

Equivalently, any codeword $\mathbf{c} \in \mathbf{C}$ satisfies

$$\mathbf{H}\mathbf{c}^{\mathrm{T}} = \mathbf{0}. \tag{2.4}$$

The above expression can also be written as

$$\sum_{i=1}^{n} \mathbf{h}_i c_i = \mathbf{0} \tag{2.5}$$

where c_i is the i^{th} symbol in $\mathbf{c}$. Equations (2.4) and (2.5) state that the syndrome of a codeword is zero.

Consider the sequence defined by

$$\sigma_0 = \mathbf{0}, \quad \sigma_1 = \mathbf{h}_1 c_1, \quad \sigma_2 = \sigma_1 + \mathbf{h}_2 c_2, \quad \ldots \quad , \sigma_n = \sigma_{n-1} + \mathbf{h}_n c_n \tag{2.6}$$

and observe that σ_n is a recursive expression describing the formation of the syndrome. A path through a trellis for a particular codeword can be traced as follows. Let the root node of the trellis correspond to $\sigma_0 = \mathbf{0}$, the all zero $n-k$ tuple. Let states at successive depths in the trellis be given by $\sigma_1, \sigma_2, \ldots, \sigma_n$. Thus the beginning and final states will be $\sigma_0 = \mathbf{0} = \sigma_n$. In principle, one can think of the trellis as being generated by considering the paths traced by all codewords. Each sequence of states in a path has a one to one correspondence with a valid codeword, since at least one of the symbols in a codeword differentiates it from all other codewords, resulting in at least one of the states in the sequence (2.6) for each codeword being different.

The formation of a state can be more concisely described using a head $\mathbf{c}^h$ of length l and the matrix $\mathbf{H}^h$, which consists of the l initial columns of $\mathbf{H}$. A state at depth l can then be written as

$$\sigma_l = \mathbf{H}^h \mathbf{c}^{h\mathrm{T}}. \tag{2.7}$$

The trellis may be formed by tracing the sequence of all states, generated by all heads $\mathbf{c}^h$ in (2.7), for each subsequent depth. Wolf's trellis construction consists of two steps. In the first step all

heads are used for all depths in (2.7), *including* those heads that do not belong to the code. This adding together of all possible symbol weighted combinations of the columns of **H** yields a state-space diagram, called the unexpurgated trellis, which represents all q^n uncoded sequences. In the second step Wolf then expurgates all paths that do not lead to the zero state at depth n, which then leaves a trellis representing the code.

As a simple example consider the binary (5,3) code with parity check matrix given by

$$\mathbf{H} = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 \end{bmatrix}. \tag{2.8}$$

The unexpurgated trellis is shown in Figure 2.1, and the final code trellis is shown in Figure 2.2.

With respect to the trellis dimensions, in [15][8] it was noted that since the state vector has length $n-k$, the number of states is at any depth is at most q^{n-k}. In other words, the maximum trellis dimension s is upper bounded by $n-k$. In the example, the maximum trellis dimension meets the upper bound of $n-k=2$.

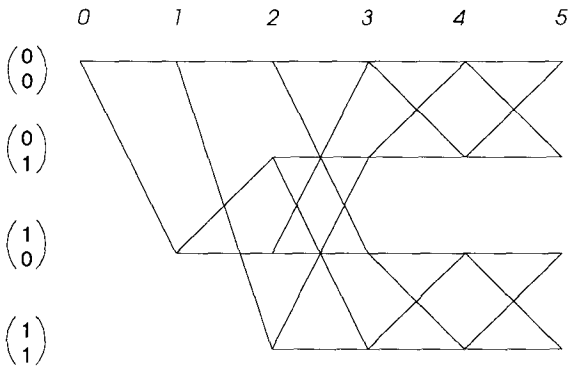


Figure 2.1. Step 1 of Wolf's Trellis Construction Method for a Binary (5,3) Code
In this step this *unexpurgated trellis* is generated. A horizontal branch corresponds to a '0' channel bit.

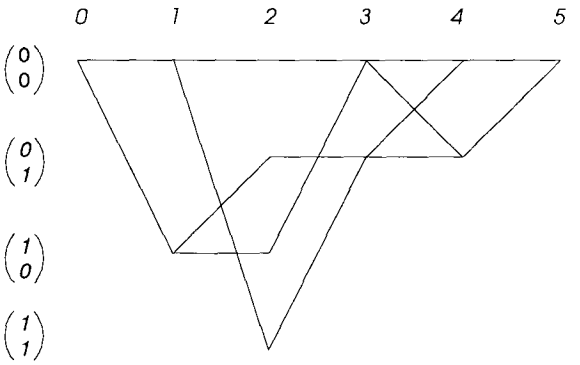


Figure 2.2. Step 2 of Wolf's Trellis Construction Method for a Binary (5,3) Code
The code trellis is obtained from the unexpurgated trellis by removing all branches that do not lead to the all zero state at depth n.

2.2.2 Massey's Trellis Construction Method

In [9] Massey discusses a trellis construction that utilizes the code's generator matrix. We will demonstrate the trellis construction method using the code of the previous example, which is also used in [9]. A generator matrix corresponding to (2.8) is

$$\mathbf{G} = \begin{array}{c} \begin{array}{ccccc} & & p_1 & & p_2 \end{array} \\ \begin{bmatrix} 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 \end{bmatrix} \\ \begin{array}{ccccc} i_1 & i_2 & & i_3 & \end{array} \end{array} \tag{2.9}$$

where we have labeled the information and parity bit positions as i_1, i_2, i_3 and p_1, p_2 respectively. We may concisely summarize Massey's state assignment method using our earlier definitions of the head and tail of a codeword. A state is assigned to be *the vector of parity bits in the tail, as determined by the information bits in the head.* Hence, a state assignment at each depth of the trellis can be written as a matrix equation using the appropriate submatrices of $\mathbf{G}$. Let us assume that $\mathbf{G}$ has been reduced to row echelon form, so that all non-pivot elements in the information-positions[9] have been eliminated. Let $\mathbf{P}$ denote a matrix formed from $\mathbf{G}$ by extracting all the columns of $\mathbf{G}$ that correspond to the parity symbols. For our example,

$$\mathbf{P} = \begin{bmatrix} 1 & 0 \\ 1 & 1 \\ 0 & 1 \end{bmatrix}. \tag{2.10}$$

Let $\mathbf{P}_j^t$ denote the submatrix obtained from $\mathbf{P}$ by retaining only the first j rows, and then of these, retaining only those columns that correspond to parity bits that are in the tail. Massey's state assignment method for a head of length l can then be expressed as

$$\sigma_l = \mathbf{i}^h \mathbf{P}_j^t \tag{2.11}$$

[9] Throughout this thesis, 'the' information positions are those independently specified positions found by reducing the generator matrix from left to right.

where $\mathbf{i}^h$ is the vector of information bits in the head, with j indicating the number of information bits in $\mathbf{i}^h$. For our example, at depth 0 we have $\mathbf{i}^h = \mathbf{0}$, $j = 0$, and so $\sigma_0 = \mathbf{0}$. At depth 1, $\mathbf{i}^h = (i_1)$, $j = 1$, and $\mathbf{P}_j^t = (1 \quad 0)$, so that states are assigned using

$$\sigma_1 = (i_1)(1 \quad 0). \tag{2.12}$$

Similarly, at depth 2 we have $\mathbf{i}^h = (i_1 \quad i_2)$, $j = 2$, and

$$\sigma_2 = (i_1 \quad i_2)\begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}. \tag{2.13}$$

At depth 3, $\mathbf{i}^h = (i_1 \quad i_2)$, $j = 2$, and a single parity bit remains in the tail, so that

$$\sigma_3 = (i_1 \quad i_2)\begin{pmatrix} 0 \\ 1 \end{pmatrix}. \tag{2.14}$$

Notice that σ_3 is a state vector of length 1, while σ_1 and σ_2 were of length 2. For convenience in drawing the trellis, we can append leading zeros to any state to form a constant length state vector of length $n - k$. At depth 4 we have,

$$\sigma_4 = (i_1 \quad i_2 \quad i_3)\begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix}. \tag{2.15}$$

Finally, at depth 5, $\sigma_0 = \mathbf{0}$. Figure 2.3 shows the trellis formed using the above equations, with leading zeros appended to state vectors of length less than $n-k$. Note that while the state assignment equations give row vectors, we will transpose them and use column vectors in indicating the states on trellis diagrams.

Massey gives a more compact trellis than that of Figure 2.3, which is shown in Figure 2.4. The more compact trellis can be found from Figure 2.3 by grouping the symbols at depths 2 and 3. (In the example in [9], Massey finds the trellis of Figure 2.4 by first grouping the symbols and then forming state equations similar to (2.12)—(2.15) .) We will concentrate on single symbol per branch trellises (as in [17]) since this allows us to easily compare different trellises for the same code.

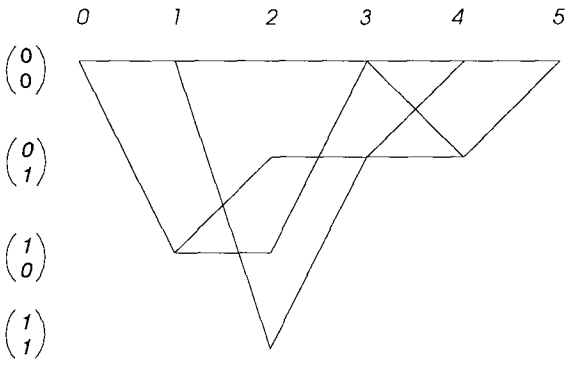


Figure 2.3. Trellis obtained using Massey's Trellis Construction Method for a (5,3) Code

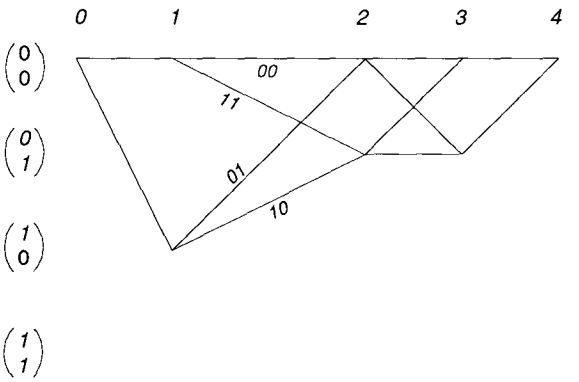


Figure 2.4. An Alternate Trellis for the (5,3) Code
This version is identical to Figure 2.3 except for a grouping of two symbols per branch for the transition from depth 1 to 2.

Comparison of the example of Massey's trellis in Figure 2.3 with the example of Wolf's trellis in Figure 2.2 reveals that they are identical. To explain why this is so, first note that in Massey's construction we began by reducing $\mathbf{G}$ to row echelon form to completely eliminate all non-pivot elements in the information positions. The remaining columns specify the generation of the parity symbols. Form a $\mathbf{H}$ matrix corresponding to $\mathbf{G}$.[10] In our example the generator matrix of (2.9) will

[10] This can be done as follows. First, since $\mathbf{G}$ has been reduced to row echelon form, and has rank k, we can extract k columns that will form an identity matrix $\mathbf{I}_k$. This leaves $n-k$ columns that form the matrix $\mathbf{P}$. From the usual discussion of systematic linear block codes, a matrix of the form $[\mathbf{I}_k \mid \mathbf{P}]$ is orthogonal to $[-\mathbf{P}^{\mathrm{T}} \mid \mathbf{I}_{n-k}]$. We can form a parity check matrix for any non-systematic code by first extracting the parity

give the corresponding parity check matrix of (2.8). Now, recall that in Wolf's trellis construction method the state assignment equation (2.7) is

$$\sigma_l^{(W)} = \mathbf{H}^h \mathbf{c}^{h\mathrm{T}} \tag{2.16}$$

where here we have used the superscript (W) to indicate that this is Wolf's state assignment. Since we require $\sigma_n = \mathbf{0}$ we must have

$$\sigma_l^{(W)} + \mathbf{H}^t \mathbf{c}^{t\mathrm{T}} = \mathbf{0} \tag{2.17}$$

where $\mathbf{H}^t$ is the tail portion of $\mathbf{H}$ (i.e. the last $n - l$ columns of $\mathbf{H}$). Note that there will exist a codeword tail, descending from this state, that has zeros in its information symbol positions. Let $\mathbf{c}'^t$ denote such a tail. Then, the state $\sigma_l^{(W)}$ in (2.16) will also satisfy

$$\sigma_l^{(W)} + \mathbf{H}^t \mathbf{c}'^{t\mathrm{T}} = \mathbf{0}. \tag{2.18}$$

The second term in (2.18) is precisely the vector of parity symbols in the tail corresponding to information symbols in the head.[11] Consequently, we can write

$$\mathbf{H}^t \mathbf{c}'^{t\mathrm{T}} = \sigma_l^{(M)\mathrm{T}} \tag{2.19}$$

where the $(M)\mathrm{T}$ superscript of σ_l indicates that the right hand side of (2.19) is equivalent to Massey's state assignment, transposed (and extended with zeros so that it has length $n-k$). Hence, using (2.19) in (2.18) we have that

$$\sigma_l^{(W)} + \sigma_l^{(M)\mathrm{T}} = \mathbf{0} \tag{2.20}$$

---

columns from $\mathbf{G}$ to form $\mathbf{P}$, then forming $-\mathbf{P}^{\mathrm{T}}$ and $\mathbf{I}_{n-k}$, and finally forming $\mathbf{H}$ by placing the $n-k$ columns of $\mathbf{I}_{n-k}$ into the parity positions (in order), and similarly placing the k columns of $-\mathbf{P}^{\mathrm{T}}$ into the information positions.

[11] To see that this is so, first consider that $\mathbf{c}'^t$ has nonzero symbols only in the parity positions of the tail, and these symbols are determined by the information symbols in the head. Next, consider that $\mathbf{H}^t$ has (by construction) columns in the parity positions that form an identity matrix, so that the product $\mathbf{H}^t \mathbf{c}'^{t\mathrm{T}}$ will simply extract out the parity symbols of $\mathbf{c}'^t$.

or

$$\sigma_l^{(M)\mathrm{T}} = -\sigma_l^{(W)}. \tag{2.21}$$

For the binary case this simplifies to

$$\sigma_l^{(M)\mathrm{T}} = \sigma_l^{(\mathrm{W})} \tag{2.22}$$

and the state assignments of Massey's and Wolf's methods are equal (except for the transpose and the extension of Massey's state vector with zeros). For the general nonbinary case the isomorphism between Massey's and Wolf's state assignments is given by (2.21).

With regard to the trellis dimensions, it is discussed in [9] that the maximum number of states at any depth of the trellis will be

$$q^{\max\left[\mathrm{rank}(\mathbf{P}_j^t)\right]}. \tag{2.23}$$

For example, from (2.12)-(2.15), the largest rank matrix occurs for depth 2, with

$$\mathbf{P}_2^t = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \tag{2.24}$$

which has rank 2, indicating that the maximum number of states is 2^2. Also, since the dimension of the trellis at each depth is the dimension of each of the $\mathbf{P}_j^t$, we have for our example the following dimensions;

$$\begin{array}{cccccccc} \text{depth} & 0 & 1 & 2 & 3 & 4 & 5 \\ s_l & 0 & 1 & 2 & 1 & 1 & 0 \end{array} \tag{2.25}$$

In closing, we emphasize that the main difference between the methods of Massey and Wolf is that Massey *directly calculates* the allowed states; instead of generating a trellis representing all uncoded sequences and then expurgating tails that do not reach the toor.[12]

[12] In [9] the state $\sigma_n = 0$ is called the *toor*, as it is the *root* node of the trellis viewed from the reverse direction.

2.2.3 Forney's Trellis Construction Method

In [16] Forney's construction of linear block code trellises utilizes a *trellis oriented generator matrix*. We briefly review his approach below, drawing upon Forney's terminology but with some alterations.

Consider the generator matrix $\mathbf{G}$ for an (n, k) linear block code $\mathbf{C}$, and an associated trellis T. The states at depths $0, 1, \ldots, n$ will be indexed by l.[13] Choose some depth l at which we divide the code's symbols into the *past* and the *future*. By the *past* we mean symbols before a node at depth l while the *future* corresponds to symbols after depth l.[14] Consider the subcode $\mathbf{C}_p$ consisting of all the codewords that are all zero in the future. Now, in [16] the *span* of a vector refers to the range of coordinates between the first and last nonzero coordinates, inclusive. Hence, the span of $\mathbf{C}_p$ is in the past. This past subcode $\mathbf{C}_p$ may be specified by a number K_p of generator matrix rows (generators). Similarly, we define the subcode $\mathbf{C}_f$ consisting of all the codewords that are all zero in the past, i.e. their span is in the future, and may be specified by K_f generators. Now, the codewords of $\mathbf{C}$ can be specified by k generators, of which we can account for $K_p + K_f$ using the past and future subcode generators. This leaves a need for $K_s = k - K_p - K_f$ generators which necessarily span parts of both the past and future[15]. We will refer to this code as the spanning code, $\mathbf{C}_s$.

One may calculate the trellis dimensions by first reducing $\mathbf{G}$ to exhibit generators for $\mathbf{C}_f$ and $\mathbf{C}_p$, then counting K_f and K_p at each depth and using $K_s = k - K_f - K_p$. Alternatively, after reducing $\mathbf{G}$ one can find K_s by inspecting the matrix to count the number of generators belonging to $\mathbf{C}_s$ at the depth. In other words, K_s will be the number of generators that cover the depth.

---

[13] We have used the term *depth*, instead of *position* as used in [16], to refer to the indexing of nodes through the trellis in order to avoid confusion with our use of position to index the n codeword coordinates.

[14] Forney's terms *past* and *future* correspond to the terms *head* and *tail*, respectively.

[15] The dimension K_s is Forney's notation for the state-space dimension at some depth, where elsewhere we have used s_l to denote the state-space dimension at depth l.

We now demonstrate Forney's method of dimension calculation and trellis construction using the example (5,3) code. A trellis oriented generator matrix for the code is

$$\begin{bmatrix} 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 \end{bmatrix} \tag{2.26}$$

from which we can determine the following dimensions;

depth	0	1	2	3	4	5
K_p	0	0	0	1	2	3
K_f	3	2	1	1	0	0
K_s	0	1	2	1	1	0

The trellis construction proceeds as follows. First, note that the trellis oriented generator matrix for some depth can be partitioned as

$$\begin{bmatrix} G_p & 0 \\ 0 & G_f \\ G_s^h & G_s^t \end{bmatrix} \tag{2.28}$$

so as to display the past and future subcode generators as well as the generators for the spanning code. The spanning code generators are shown as being split into their head and tail portions. Forney's method labels states at a depth with all the codewords generated by the heads of the spanning code, G_s^h.

For our example, we obtain the following state assignment equations, by reading off G_s^h for each depth from (2.26)

$$\sigma_0 = \mathbf{0} \tag{2.29}$$

$$\sigma_1 = (i_1)(1) \tag{2.30}$$

$$\sigma_2 = (\, i_1 \quad i_2 \,)\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \tag{2.31}$$

$$\sigma_3 = (i_2)(1 \quad 1) \tag{2.32}$$

$$\sigma_4 = (i_3)(0 \quad 1) \tag{2.33}$$

$$\sigma_5 = \mathbf{0} \tag{2.34}$$

These equations generate the trellis shown in Figure 2.5.

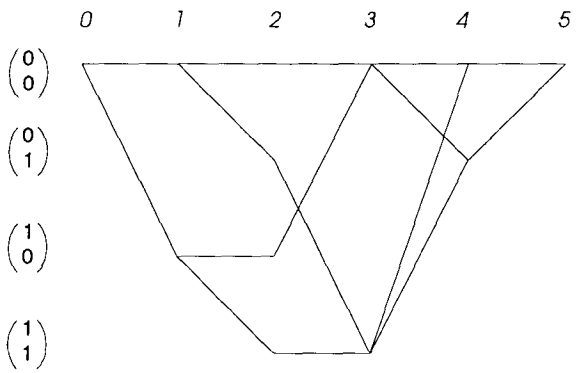


Figure 2.5. Trellis obtained using Forney's Trellis Construction Method for a (5,3) Code

2.3 Minimal Trellises

We are interested in trellises that have the fewest number of states. In [17], Muder defines a trellis T for a code $\mathbf{C}$ to be *minimal* if for every trellis T' for $\mathbf{C}$ the number of states at each depth is minimal, i.e.

$$|V_l| \leq |V_l'| \qquad \forall l. \tag{2.35}$$

Muder showed that minimal trellises exist for every linear block code as well as determining the conditions for minimality. The approach was to begin with a tree $\hat{T}$ and then view trellis states as representing heads in $\hat{T}$ that satisfy a certain equivalence relation. For a trellis to be minimal, it is necessary and sufficient that a state at depth l be assigned to those heads $\mathbf{c}^h$ of the tree $\hat{T}$ that satisfy the equivalence relation

$$\mathbf{c}^h \sim_t \mathbf{c}'^h \tag{2.36}$$

where $\sim_t$ indicates that $\mathbf{c}^h$ and $\mathbf{c}'^h$ share the same set of tails. The equivalence classes resulting from the partitioning of heads by $\sim_t$ at depth l form the set of states V_l for a minimal trellis. All minimal trellises for a given code are isomorphic [17, Theorem 1].

In [17], Muder used this minimality condition (i.e. that minimal trellises have states assigned according to the $\sim_t$ equivalence classes) to show that Forney's construction method yields a minimal trellis. Here, we use Muder's minimality condition to prove the following proposition.

Proposition 2.1: Wolf's method [8] for constructing trellises for general linear block codes, and Massey's method [9], yield *minimal* trellises.

Proof: Since we have shown earlier that Massey's trellis construction method results in a trellis that is isomorphic to the trellis generated by Wolf's method, it is sufficient to show that Wolf's method results in a minimal trellis.

If an $(n-l)$–tuple $\mathbf{c}^t$ is to be a tail of $\mathbf{c}^h$ then we must have from $\mathbf{H}\mathbf{c}^{\mathrm{T}} = \mathbf{0}$ that

$$\mathbf{H}^h \mathbf{c}^{h\mathrm{T}} = -\mathbf{H}^t \mathbf{c}^{t\mathrm{T}}. \tag{2.37}$$

Equation (2.37) implies that for two heads $\mathbf{c}^h$ and $\mathbf{c}'^h$ to share a tail we require that

$$\mathbf{H}^h \mathbf{c}^{h\mathrm{T}} = \mathbf{H}^h \mathbf{c}'^{h\mathrm{T}}. \tag{2.38}$$

Also, heads that satisfy (2.38) and (2.37) will share all of their tails. In other words, heads that satisfy (2.38) and (2.37) will belong to an equivalence class defined by $\sim_t$. From Muder's condition that a minimal trellis can be constructed by assigning a state to those heads in a $\sim_t$ equivalence class, we can construct a minimal trellis by assigning a state to those heads that satisfy (2.38) and (2.37). However, this is precisely the state assignment definition (2.7) used in Wolf's trellis construction method.

2.4 Simplified Trellis Construction Methods

We now describe three simplified methods of constructing a minimal trellis for general linear block codes. The methods avoid the generation/expurgation of heads using Wolf's unexpurgated trellis, and also avoid matrix multiplication at each state computation as in Massey's and Forney's methods. Some other features also contribute to their simplicity, which will be summarized at the end of this section.

2.4.1 Method 1

Method 1 assigns each state to be a vector of parity check symbols, just as in [15] [8]. The resulting trellis will be isomorphic to these trellises, and so will be a minimal trellis. Our presentation

of the trellis construction method begins with finding an alternative method for assessing the trellis dimensions. This method of assessing the trellis dimensions will give identical results to Massey's and Forney's methods since all of these trellises are minimal.

The parity check matrix type trellises of [15] [8] assigns as a state to a head the vector (partial syndrome) formed by summing columns of $\mathbf{H}$ weighted by the corresponding head symbols. To find the trellis dimensions we will consider the dimension of the column space of partitions of $\mathbf{H}$, where a partition of $\mathbf{H}$ consists of head and tail submatrices. The head submatrix $\mathbf{H}^h$ contains the first l columns of $\mathbf{H}$ and the tail submatrix $\mathbf{H}^t$ contains the remaining $n - l$ columns.

Elementary row operations on $\mathbf{H}$ will not change the solution to $\mathbf{H}\mathbf{c}^{\mathrm{T}} = \mathbf{0}$, so that we are free to construct trellises from such transformed parity check matrices. Although any set of elementary row operations on $\mathbf{H}$ will preserve the code, here we prefer to put $\mathbf{H}$ into a standard form, denoted $\mathbf{H}_{std}$, for reasons that will be discussed later. The actions to be performed on $\mathbf{H}$ to reduce it to standard form are summarized in Figure 2.6. First, Step 1 uses elementary row operations to reduce $\mathbf{H}$ to lower trapezoidal form, with the elimination proceeding from right to left and bottom to top. Since the rank of $\mathbf{H}$ is $n - k$ this reduction will yield $n - k$ *tail pivots*, whose column positions are indicated in Figure 2.6 by downward pointing arrows along the top of the matrix. For any head/tail partition of $\mathbf{H}$, this step will identify tail pivots with corresponding columns that can be taken as basis vectors for the column space of $\mathbf{H}^t$. (This follows from the fact that pivots found by row reduction also correspond to the pivot locations found if column reduction was used [21].) The $\mathbf{H}$ matrix after Step 1 will be denoted $\mathbf{H}_1$. The second step is to use elementary row operations, *except row exchanges*, to find $n - k$ *head pivots*, by proceeding from left to right and from top to bottom. The resulting matrix is the desired $\mathbf{H}_{std}$, and the column positions of the head pivots are indicated in Figure 2.6 by upward pointing arrows along the bottom of the matrix. For any partition of $\mathbf{H}$ the

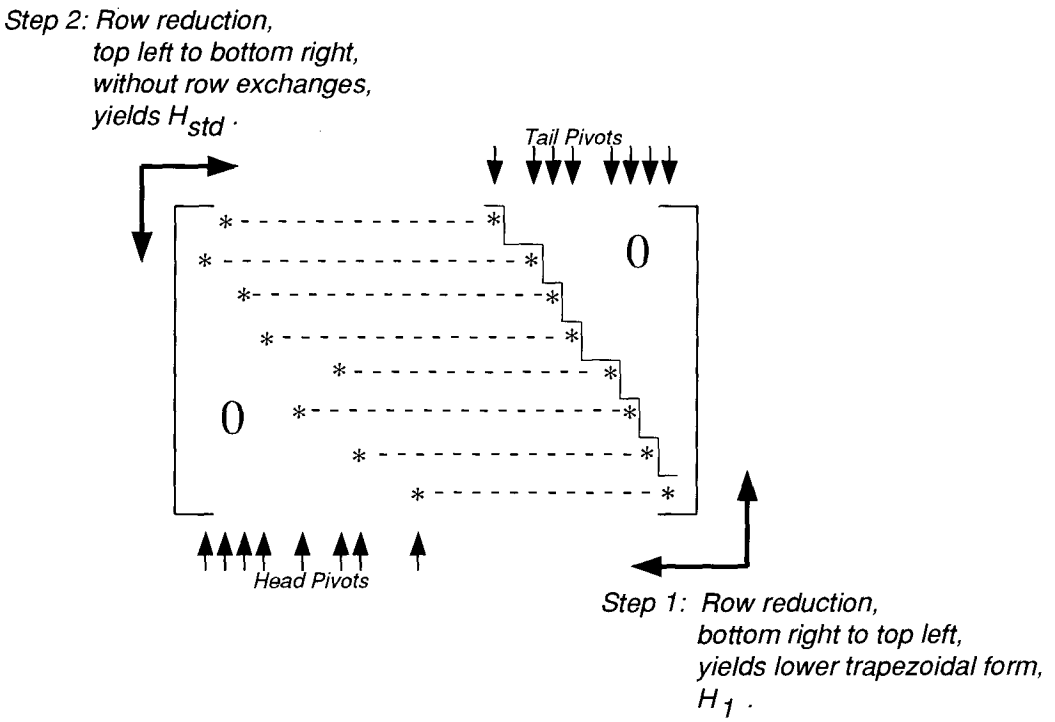


Figure 2.6 Reduction of Parity Check Matrix to Standard Form.
The *'s represent the row and column position of the pivots.

head pivots identify those columns that may be taken as basis vectors for the column space of $\mathbf{H}^h$.

Now consider the formation of the trellis using $\mathbf{H}_{std}$. Any codeword satisfies (2.5), or equivalently

$$\mathbf{H}^h \mathbf{c}^{h\mathrm{T}} + \mathbf{H}^t \mathbf{c}^{t\mathrm{T}} = \mathbf{0} \tag{2.39}$$

where, in order to simplify the notation, we have used $\mathbf{H}$ for $\mathbf{H}_{std}$. We may rewrite (2.39) as

$$\sigma_l + \mathbf{H}^t \mathbf{c}^{t\mathrm{T}} = \mathbf{0} \tag{2.40}$$

where σ_l is the state for $\mathbf{c}^h$ at depth l. Consider an extension from depth l to depth $l+1$, with the new state given by

$$\sigma_{l+1} = \sigma_l + \mathbf{h}_{l+1} c_{l+1}. \tag{2.41}$$

Proposition 2.2: Let I_h and I_t denote the set of symbol positions at which head and tail pivots, respectively, are found in $\mathbf{H}$. Then the increase in trellis state-space dimension after an extension from depth l to $l+1$ is

$$s_{l+1} - s_l = \begin{cases} 0 & l+1 \notin I_h, \ l+1 \notin I_t \\ 1 & l+1 \in I_h, \ l+1 \notin I_t \\ -1 & l+1 \notin I_h, \ l+1 \in I_t \\ 0 & l+1 \in I_h, \ l+1 \in I_t \end{cases} \tag{2.42}$$

Proof: Let $\mathbf{H}^{h(ext)}$ denote $\mathbf{H}^h$ after extension of the heads to depth $l+1$. Similarly, let $\mathbf{H}^{t(ext)}$ denote $\mathbf{H}^t$ after extension. During extension, the column $\mathbf{h}_{l+1}$ is transferred from $\mathbf{H}^t$ to $\mathbf{H}^{h(ext)}$. For any state at depth $l+1$, we must have from (2.40) that

$$\sigma_{l+1} + \mathbf{H}^{t(ext)} \mathbf{c}^{t(ext)\mathrm{T}} = \mathbf{0} \tag{2.43}$$

where $\mathbf{c}^{t(ext)}$ the codeword tail for $\mathbf{c}$ after extension of the head to depth $l+1$. Since (2.43) expresses an linear dependence condition it must hold that the state-space of the heads at depth $l+1$ is spanned by the columns of $\mathbf{H}^{t(ext)}$. We consider the four cases in (2.42) in turn.

Case (i) $l+1 \notin I_h, \quad l+1 \notin I_t$. With $l+1 \notin I_h$, then the state-space dimension cannot be increased, since $\mathbf{h}_{l+1}$ is not linearly independent of $\mathbf{H}^h$. Also, with $l+1 \notin I_t$ the columns of $\mathbf{H}^{t(ext)}$ span the same space as those of $\mathbf{H}^t$, so that any state in existence at depth l will also satisfy (2.43). Hence $s_{l+1} - s_l = 0$.

Case (ii) $l+1 \in I_h, \ l+1 \notin I_t$. With $l+1 \notin I_t$ we have from Case (i) that all states in existence at depth l will also satisfy (2.43). With $l+1 \in I_h$ the state-space dimension will increase by one, provided all such states satisfy (2.43). This will be true since the transferral of $\mathbf{h}_{l+1}$ from

$\mathbf{H}^t$ to $\mathbf{H}^{h(ext)}$ does not change the span of $\mathbf{H}^{t(ext)}$, thus the inclusion of $\mathbf{h}_{l+1}$ in $\mathbf{H}^{h(ext)}$ adds a dimension without violating (2.43).

Case (iii) $l+1 \notin I_h, \quad l+1 \in I_t$. As in Case (i), with $l+1 \notin I_h$ the state-space dimension cannot be increased. Assume that the state space dimension remains constant. This would require that $\mathbf{H}^{t(ext)}$ span the same space as $\mathbf{H}^t$. However, this is not the case since $\mathbf{h}_{l+1}$ is not in the span of $\mathbf{H}^{t(ext)}$, so that the state space dimension must be reduced by one.

Case (iv) $l+1 \in I_h, \quad l+1 \in I_t$. Here the effect of the transferral of $\mathbf{h}_{l+1}$ may be viewed as increasing the dimension of the extended state–space due to the fact that $\mathbf{h}_{l+1}$ is a basis vector for $\mathbf{H}^{h(ext)}$, followed by a decrease in dimension due to the fact that $\mathbf{h}_{l+1}$, while being in the span of $\mathbf{H}^{h(ext)}$, is not in the span of $\mathbf{H}^{t(ext)}$. Hence the result is no net change in the state-space dimensionality. ———————————

Evaluating the trellis dimensions from a parity check matrix in standard form using Proposition 2.2 is particularly simple. One needs to simply increment the dimension by one for each head pivot and similarly decrement it for each tail pivot. As an example, we return to our (5,3) code example, with its parity check matrix;

$$\begin{array}{ccccccc} & & & \downarrow & & \downarrow & \\ \left[\right. & 1 & 1 & 1 & 0 & 0 & \left.\right] \\ \left[\right. & 0 & 1 & 0 & 1 & 1 & \left.\right] \\ & \uparrow & \uparrow & & & & \end{array} \tag{2.44}$$

which happens to already be in standard form. We have indicated the positions of the head and tail pivots with up and down arrows, respectively. Beginning at depth 0 with $s_0 = 0$ and working to depth 5, we increment s_l for each 'up' arrow and decrement it for each 'down' arrow, to obtain ;

$$\begin{array}{ccccccc} \text{depth} & 0 & 1 & 2 & 3 & 4 & 5 \\ s_l & 0 & 1 & 2 & 1 & 1 & 0 \end{array} \tag{2.45}$$

For the purpose of finding the trellis dimensions, it is sufficient to have $\mathbf{H}$ in standard form, but it is not necessary. All that is required to use the increment/decrement rule of Proposition 2.2 is to have reduced the parity check matrix to expose the head and tail pivots.

By viewing $\mathbf{H}$ as the generator matrix for $\mathbf{C}^{\perp}$ (the dual code to $\mathbf{C}$) the reduction of $\mathbf{H}$ to expose the head and tail pivots will yield a trellis oriented generator matrix for $\mathbf{C}^{\perp}$. Moreover, since Forney showed that the state-space dimensions for $\mathbf{C}$ or its dual $\mathbf{C}^{\perp}$ are identical [16] we can compute the trellis dimensions beginning with either $\mathbf{G}$ or $\mathbf{H}$, then reducing (which finds a matrix which can be considered to be a trellis oriented generator matrix or a parity check matrix with head/tail pivots exposed), and then finally computing the trellis dimensions by counting the number of spanning generators or by incrementing/decrementing a dimension 'counter' according to the positions of the head/tail pivots. We comment that the increment/decrement method of Proposition 2.2 is somewhat simpler, since at each depth one needs only to 'bump' a counter after inspection of 2 items (the up and down arrows that indicate the pivot positions); as opposed to inspecting k items (the rows of $\mathbf{G}$) and counting up all those that meet a test of whether they belong to the spanning code.

Trellis Construction: Method 1.

A minimal trellis can be constructed in a breadth-first manner and without the codeword expurgation used by Wolf [8], as described below. The parity check matrix to be used can be $\mathbf{H}_1$ or $\mathbf{H}_{std}$, or any $\mathbf{H}$ matrix that has been row reduced from 'right to left' to find tail pivots. We will refer to a position at which there is a tail pivot in the reduced matrix as a *constraint* position.

Trellis Construction (Method 1)

1. ***Initialization:*** Set $l = 0$, $\sigma_0 = \mathbf{0}$, $V_0 = \{\sigma_0\}$ so that at depth 0 the set of states V_0 consists only of a root.

2. ***Constraint Position Test:*** If the position $l+1 \notin I_t$ the extension is unconstrained; perform step 3. If the position $l+1 \in I_t$ the extension is constrained; perform step 4.
3. ***Unconstrained Extension:*** For all states σ_l in V_l, p%%[Error: timeout; OffendingCommand

$$\sigma_{l+1} = \sigma_l + \alpha_j \mathbf{h}_{l+1} \qquad , j = 0, 1, \ldots, q-1 \tag{2.46}$$

and label each branch from σ_l to σ_{l+1} with its associated symbol α_j. (This extends each state at depth l with q branches to states at depth $l+1$.) Perform step 5.

4. ***Constrained Extension:*** For each state σ_l in V_l; Let α_c denote the value of the state vector element which corresponds to the row of the tail pivot at position l. Let $\alpha_c' = -\alpha_c$ denote the additive inverse of α_c. Set

$$\sigma_{l+1} = \sigma_l + \alpha_c' \mathbf{h}_{l+1} \tag{2.47}$$

and label the branch from σ_l to σ_{l+1} with α_c'. (This extends each state at depth l with only a single branch.)

5. ***Termination Test:*** If $l = n-1$ the trellis is complete. Otherwise increment l and perform step 2.

Using parity check matrices that have been reduced to lower trapezoidal form, such as $\mathbf{H}_1$ or $\mathbf{H}_{std}$, offers a minor advantage in that the state extensions at each constraint position become slightly easier to construct. A pointer initialized to 0 and incremented at each constraint position can be used to pick off α_c from the state σ_l. The resulting trellis will have the characteristic that at each successive constraint position the remaining leading element of the state vector is disallowed (i.e. must be zero). Using parity check matrices that have been reduced to standard form allows the trellis dimensions to be computed prior to trellis construction.

As an example, consider a Hamming (7,4) code specified by the parity check matrix

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

$$0 \quad 1 \quad 2 \quad 3 \quad 2 \quad 2 \quad 1 \quad 0$$

which can be seen to have been obtained by interchanging positions 1 and 5 of a systematic parity check matrix. In this case, the matrix is already in standard form, and we have indicated the head and tail pivots along with the resulting trellis dimensions. The corresponding trellis appears in Figure 2.7 which shows the leading ('most significant') bits of the state vector being successively disallowed at each successive constraint depth.

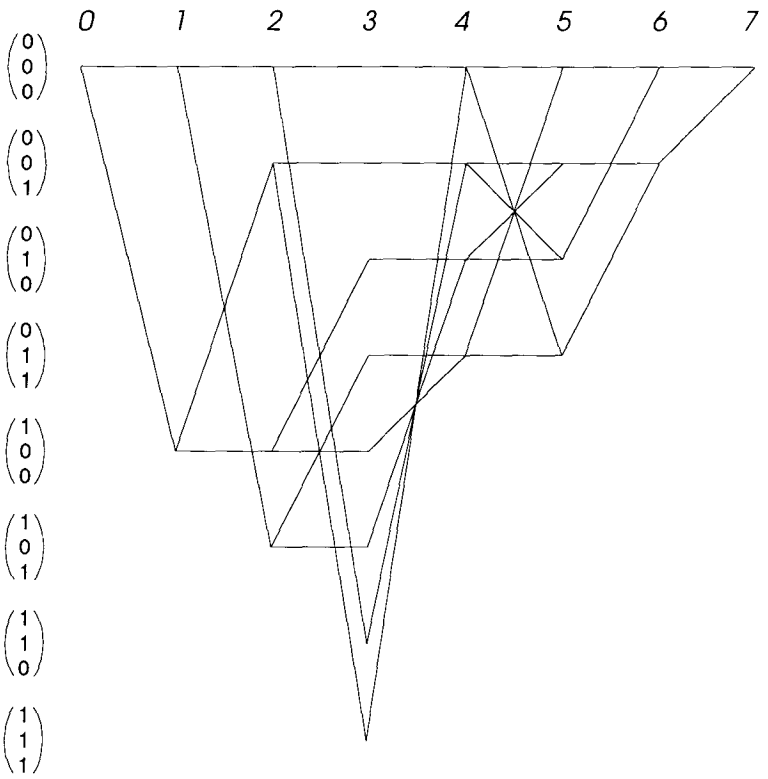


Figure 2.7 Trellis for a (7,4) Hamming Code Generated using Method 1
The trellis was generated from a parity check matrix in lower trapezoidal form, which results in the most significant bits of the state vector being successively disallowed at each constraint depth.

2.4.2 Method 2

Method 2 is a refinement of Method 1, in that the state extensions are even simpler to compute. This simplification of the state computations occurs at constraint positions.

For Method 2, the parity check matrix is reduced to row echelon form, so that each pivot is the only nonzero element in its column. We again require that the reduction of $\mathbf{H}$ be carried out from right to left and from bottom to top. The resulting form will be similar to that of $\mathbf{H}_1$ as shown in Figure 2.6, except that complete reduction is used to eliminate all nonpivot elements. Let the resulting matrix be denoted as $\mathbf{H}_2$. For the example parity check matrix on page 33, we obtain $\mathbf{H}_2$ as

$$\begin{bmatrix} 1 & 1 & 1 & \overset{\downarrow}{1} & 0 & \overset{\downarrow}{0} & \overset{\downarrow}{0} \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix} \tag{2.48}$$

In the following algorithm, steps 1 and 2 are slightly modified versions of the respective steps in Method 1, and steps 3 and 5 are identical to the respective steps in Method 1. The main change of Method 2 with respect to Method 1 is in step 4.

Trellis Construction (Method 2)

1. ***Initialization:*** Set $l = 0$, $\sigma_0 = \mathbf{0}$, $V_0 = \{\sigma_0\}$ so that at depth 0 the set of states V_0 consists only of a root. Initialize a counter i_P to 0. (The counter i_P will be used to point to successive elements of the state vector.)
2. ***Constraint Position Test:*** If the position $l+1 \notin I_t$ the extension is unconstrained; perform step 3. If the position $l+1 \in I_t$ the extension is constrained; increment i_P and perform step 4.
3. ***Unconstrained Extension:*** For all states σ_l in V_l, perform

$$\sigma_{l+1} = \sigma_l + \alpha_j \mathbf{h}_{l+1} \qquad , j = 0, 1, \ldots, q-1 \tag{2.49}$$

and label each branch from σ_l to σ_{l+1} with its associated symbol α_j. (This extends each state at depth l with q branches to states at depth $l+1$.) Perform step 5.

4. ***Constrained Extension:*** For each state σ_l in V_l; Form σ_{l+1} by setting the i_P^{th} element of σ_l to zero. Let α_c denote the value of the i_P^{th} element in the state vector σ_l, and let $\alpha_c' = -\alpha_c$ denote the additive inverse of α_c. Label the branch from σ_l to σ_{l+1} with α_c'. (This extends each state at depth l with only a single branch.)
5. ***Termination Test:*** If $l = n-1$ the trellis is complete. Otherwise increment l and perform step 2.

Method 2 makes explicit use of a counter (i_P) to point to successive elements of the state vector at each constraint depth. More importantly, the row echelon form of $\mathbf{H}_2$ is exploited to simplify the next-state formation at a constraint depth. The state vector element pointed to by i_p is simply set to zero. Figure 2.8 shows the resulting trellis.

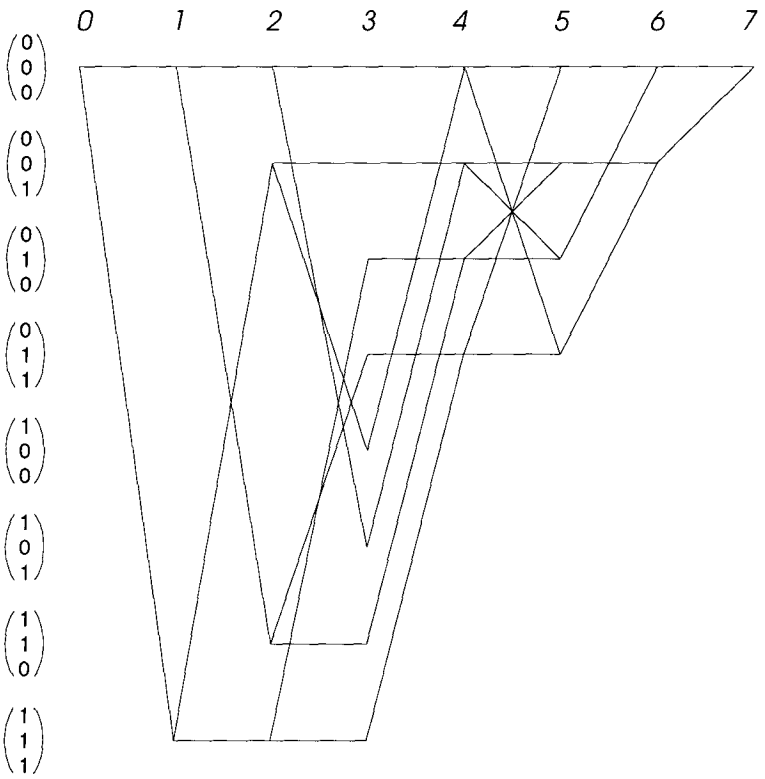


Figure 2.8 Trellis for a (7,4) Hamming Code Generated using Method 2
The trellis was generated from a parity check matrix in a modified row echelon form, which facilitates the next-state computation at constraint positions. At each constraint-position it can be seen that the next-state is formed from the present state by simply setting the remaining most significant bit of the state vector to zero.

2.4.3 Method 3

Method 3 is a simple modification of Massey's trellis construction method for non-systematic linear block codes [9]. Method 3 is a simplification of the method of [9] in that it avoids matrix multiplication to extend each state to the next depth. As in [9], we begin by reducing $\mathbf{G}$ to row echelon form, to completely eliminate all non-pivot elements. As discussed earlier, let $\mathbf{P}$ denote the matrix formed by extracting the $n-k$ columns of $\mathbf{G}$ that correspond to the parity positions. Also, let $\mathbf{p}_m$ denote the m^{th} row of $\mathbf{P}$, and let $\mathbf{p}_m^t$ denote the tail of the m^{th} row of $\mathbf{P}$ (i.e. $\mathbf{p}_m^t$ consists of the elements in the m^{th} row of $\mathbf{P}$ that correspond to positions in the codeword tail.). In the following algorithm, steps 1 and 2 are similar to the respective steps in Method 2, while step 5 is identical. The main differences from Method 2 are in steps 3 and 4.

Trellis Construction (Method 3)

1. ***Initialization:*** Set $l = 0$, $\sigma_0 = \mathbf{0}$, $V_0 = \{\sigma_0\}$ so that at depth 0 the set of states V_0 consists only of the root. Initialize an information-symbol counter i_I to zero. Initialize a parity-symbol counter i_P to zero.
2. ***Constraint Position Test:*** If the position $l+1 \notin I_t$ the extension is unconstrained; increment i_I and perform step 3. If the position $l+1 \in I_t$ the extension is constrained; increment i_P and perform step 4.
3. ***Unconstrained Extension:*** For all states σ_l in V_l, perform

$$\sigma_{l+1} = \sigma_l + \alpha_j \mathbf{p}_{i_I}^t \qquad , j = 0, 1, \ldots, q-1 \tag{2.50}$$

and label each branch from σ_l to σ_{l+1} with its associated symbol α_j. Here $\mathbf{p}_{i_I}^t$ is the tail portion (i.e. the last $n-k-i_P$ elements) of the i_I^{th} row of $\mathbf{P}$. (This step extends each state at depth l with q branches to states at depth $l+1$.) Perform step 5.

4. ***Constrained Extension:*** For each state σ_l in V_l; Form σ_{l+1} by setting the i_P^{th} element of σ_l to zero. Label the branch from σ_l to σ_{l+1} with the i_P^{th} element of σ_l. (This extends each state at depth l with only a single branch.)
5. ***Termination:*** If $l = n - 1$ the trellis is complete. Otherwise increment l and perform step 2.

2.4.4 Discussion

Method 1 utilizes any form of the parity check matrix that has been row reduced from 'right to left' to find tail pivots. In particular, Method 1 can be used with a parity check matrix in 'standard' form,[16] which facilitates the computation of the trellis dimensions as well as the trellis construction.

Method 2 is a specialized version of Method 1, and uses the parity check matrix reduced to modified row echelon form. While Method 2 does not directly provide the trellis dimensions, it greatly simplifies the next-state formation at constraint-positions. Indeed, there need not be *any* modification made to state vectors when extending them to constraint positions, if we agree to mask off (ignore) each successive leading bit of the state vector as indicated by the counter i_P. This can be done since each state at a constraint depth is known to have a zero element in the position indicated by the counter i_P.

Method 3 utilizes the generator matrix reduced to row echelon form, and its state assignment requires similar effort to that of Method 2. Branch labeling at constraint positions is perhaps simplest in Method 3 since the branch label is simply taken to be the value of a particular element of the state vector, while in Method 2 the label is the additive inverse of the same. However, this minor advantage does not exist for binary codes.

[16] The 'standard form' parity check matrix $\mathbf{H}_{std}$ is equivalent to Forney's 'trellis oriented generator matrix', except for a specified ordering of the rows to have the matrix in lower trapezoidal form.

We emphasize that each of Methods 1, 2, and 3, share the following principal advantages over previous methods. They avoid the generation of an unexpurgated trellis (followed by expurgation of invalid tails) as used in [8] for general linear block codes. They also avoid matrix multiplication at each extension to form the trellis, as used in [9] for non-systematic linear block codes, or as used in [16] for general linear block codes.

The simplified trellis construction methods can also be used to construct and search partial trellises. As will be discussed in subsequent chapters, partial trellis or tree searches are useful where the full trellis is so large that the storage, generation, or searching of the entire trellis is unfeasible. The simplified trellis construction methods presented here should be useful in the construction of partial trellises 'on the fly', as guided by the search algorithm, so that only the portion of the trellis that needs to be explored is generated.

2.5 Bounds on Trellis Dimensions

Since the number of states in the trellis affects the ML decoding effort it is not only of interest to construct a minimal trellis but also to have simple bounds on the number of trellis states attainable for different code parameters (n, k, d_{min}) [17]. A complication arises in that the trellis state space dimensions can be altered by a permutation of symbol positions, as pointed out by Massey [9] and others, including; Bahl *et al*[15], Matis and Modestino [22], Muder[17], and Kasami *et al* [23].

Bahl *et al* [15] and Wolf [8] established a simple upper bound on the maximum trellis dimension. Muder [17] introduced some lower bounds on the maximum trellis dimension of a minimal trellis. Herein we improve on Muder's bound for general linear block codes. As well we show that the variability of trellis dimensions due to symbol position permutation is limited to a central region of

the trellis. The dimensions in the fixed region are determined and we give a simple lower bound on the minimum trellis dimensions in the variable dimension region.

In [15] [8] it was noted that a parity check matrix based trellis will have $s \leq n-k$, which follows from the fact that there are $n-k$ symbols in the state vector. Also, since there are q^k codewords, or using the fact that the state-space dimensions of the dual code $\mathrm{C}^{\perp}$ (with k parity check symbols) are identical to C [16], we have also that $s \leq k$. These upper bounds are summarized as

$$s \leq \min(k, n-k). \tag{2.51}$$

The first new result of this section is the following proposition.

Proposition 2.3: For any linear block code C the state-space dimensions at the d consecutive depths at either end of the trellis are

$$s_l = \begin{cases} l & l \in \{0, 1, 2, \ldots, d-1\} \\ n-l & l \in \{n-(d-1), \ldots, n-1, n\} \end{cases} \tag{2.52}$$

for any symbol position permutation, where

$$d \triangleq \min\left(d_{min}, d_{min}^{\perp}\right) \tag{2.53}$$

is the smaller of the minimum distances of the code C or its dual $\mathrm{C}^{\perp}$.

Proof: We first show that $s_l = l,\ l \in \{0, 1, 2, \ldots, d-1\}$. For this, from Proposition 2.2 it is sufficient to show that $l \in I_h$ and $l \notin I_t$ for $l \in \{1, 2, \ldots, d-1\}$. The fact that any $d_{min}-1$ columns of $\mathbf{H}$ are linearly independent implies that there will be $d_{min}-1$ head pivots in positions $1, 2, \ldots, d_{min}-1$, so we have $l \in I_h$ for $l \in \{1, 2, \ldots, d_{min}-1\}$. The minimum weight of any row in $\mathbf{H}$ is $d_{min}^{\perp}$, since $\mathbf{H} = \mathbf{G}^{\perp}$. The earliest that a tail pivot can possibly occur is for the case where a minimum weight row has all of its nonzero symbols bunched together in the earliest positions. This corresponds to a weight $d_{min}^{\perp}$ row having its $d_{min}^{\perp}$ nonzero symbols in the first $d_{min}^{\perp}$

positions. Consequently, the earliest that a tail pivot can occur is at depth $d_{min}^{\perp}$, so that $l \notin I_t$ for $l \in \{1, 2, \ldots, d_{min}^{\perp} - 1\}$. This fact plus the fact that $l \in I_h$ for $l \in \{1, 2, \ldots, d_{min} - 1\}$ implies that $l \in I_h$, $l \notin I_t$ for $l \in \{1, 2, \ldots, d-1\}$, where $d \triangleq \min\left(d_{min}, d_{min}^{\perp}\right)$. Finally, to establish that $s_l = n - l$ for $l \in \{n-(d-1), \ldots, n-1, n\}$ we observe that the above arguments hold when viewing the parity check matrix in the reverse direction from depth n. ———————————

Proposition 2.3 provides us with a lower bound on the minimum trellis dimension as follows.

Corollary 2.3.1: For any linear block code the maximum trellis dimension satisfies

$$s \geq \min\left(d_{min} - 1, d_{min}^{\perp} - 1\right) \tag{2.54}$$

or equivalently,

$$s \geq d - 1. \tag{2.55}$$

Proof: For a code and its dual with smallest minimum distance d, Proposition 2.3 indicates that the trellis dimension at depth $d-1$ will always be $d-1$, so that the maximum trellis dimension is at least this value. ———————————

Corollary 2.3.2: If C is a maximal distance separable (MDS) code, then

$$s = \min(k, n-k). \tag{2.56}$$

Proof: For an MDS code $d_{min} - 1 = n - k$ so that any $n-k$ codeword positions are parity positions and any k positions form an information set. (In other words, any k codeword positions are independently specified, and the remainder are forced by the constraints of the code.) Now,

viewing $\mathbf{H}$ as the generator matrix for $\mathbf{C}^{\perp}$, we have that any $n-k$ positions are independently specified and hence any k positions form a parity set, i.e. any k columns of $\mathbf{H}^{\perp}$ will be linearly independent. Hence $d_{min}^{\perp}-1=k$ and $\mathbf{C}^{\perp}$ is MDS. So, using $d_{min}-1=n-k$ and $d_{min}^{\perp}-1=k$ in Corollary 2.4.1 we have $s \geq \min(k, n-k)$. This, combined with the fact that $s \leq \min(k, n-k)$ establishes (2.56). ∎

That MDS codes have a maximum trellis dimension given by (2.56) was shown by Muder [17] by bounding the dimensions of the past and future subcodes. Using the same method Muder obtained the following lower bounds on s,

$$\begin{aligned} s &\geq \min(k, n-k-2\Delta) \\ s &\geq \min\left(n-k, k-2\Delta^{\perp}\right) \end{aligned} \tag{2.57}$$

where

$$\Delta \triangleq n-k-(d_{min}-1) \tag{2.58}$$

and

$$\Delta^{\perp} \triangleq k-\left(d_{min}^{\perp}-1\right). \tag{2.59}$$

To see that Corollary 2.3.1 improves on (2.57), we simplify (2.57) as follows

$$\begin{aligned} s &\geq \min\left(k, n-k-2\Delta, n-k, k-2\Delta^{\perp}\right) \\ &= \min\left(n-k-2\Delta, k-2\Delta^{\perp}\right) \\ &= \min\left(d_{min}-1-\Delta, d_{min}^{\perp}-1-\Delta^{\perp}\right) \end{aligned} \tag{2.60}$$

where we have used the fact that $\Delta \geq 0$ and that $\Delta^{\perp} \geq 0$. Comparing the equivalent form (2.60) of Muder's bounds (2.57) to Corollary 2.3.1 (2.54) reveals our improvement in that the terms Δ and $\Delta^{\perp}$ have been eliminated.

For example, the Golay (24,12,8) code has $\Delta = \Delta^{\perp} = 5$ and Muder's bound of (2.57) yields $s \geq 2$, while Proposition 2.3 gives $s \geq 7$. This is significantly closer to the known results for the smallest s possible for this code, which is 9 [17].

Proposition 2.3 indicates that the trellis dimensions for the first $d-1$ consecutive depths at either end of the trellis do not vary regardless of symbol position permutation. We will refer to these depths, $l < d, l > n-d$, as *fixed dimension regions*. Between these fixed dimension regions the trellis dimensions may vary with symbol position permutation, and we refer to these depths $d \leq h \leq n-d$ as the *variable dimension region*. We now lower bound the minimum dimension of the variable dimension region,

$$s' \triangleq \min_{(\forall \mathbf{P})\mathbf{G}} \left[\min_{d \leq h \leq n-d} (s_h) \right] \tag{2.61}$$

where $\mathbf{P}$ denotes a permutation matrix. Hence $(\forall \mathbf{P})\mathbf{G}$ generates all permutations of symbol positions of $\mathbf{G}$ and the corresponding equivalent codes.

Proposition 2.4: A lower bound on the minimum dimension of the variable dimension region is

$$s' \geq \max \left[0, \min \left(d_{min} - 1 - \Delta, d_{min}^{\perp} - 1 - \Delta^{\perp} \right) \right]. \tag{2.62}$$

Proof: Consider a (n, k, d_{min}) code $\mathbf{C}$. Either we will have $d = d_{min}$ or $d = d_{min}^{\perp}$. First, assume that $d = d_{min}$. From Proposition 2.4 we have that the trellis dimensions are $d_{min} - 1$ at depth $d_{min} - 1$, independent of any permutation of symbol positions. From Proposition 2.2 recall that a decrease in dimension is possible at a tail pivot, of which there are $n-k$. Now $d_{min}-1$ of these are confined to consecutive positions at the toor, so there are $\Delta = n - k - (d_{min} - 1)$ tail pivots which may change positions for equivalent codes obtained by symbol position permutation. Hence, a lower bound on the smallest state-space dimension would be if all Δ tail pivots followed position $d_{min}-1$,

before any remaining head pivots, resulting in $s' \geq d_{min} - 1 - \Delta$. Now assume that $d = d^{\perp}_{min}$, so that we consider the code $\mathrm{C}^{\perp}$ with minimum distance $d^{\perp}_{min}$ and k pivots. Repeating the argument used above for the code C, we have that $s = d^{\perp}_{min} - 1$ at depth $d^{\perp}_{min} - 1$, and $s' \geq d^{\perp}_{min} - 1 - \Delta^{\perp}$. These two cases and the fact that $s \geq 0$ give Proposition 2.4. ———————

Note that Proposition 2.4 may also be used to show that MDS codes have $s = \min(k, n-k)$, since $\Delta = \Delta^{\perp} = 0$ and the lower bound on the minimum state-space dimension (2.62) will agree with the upper bound of (2.51).

In light of Proposition 2.4 we revisit Muder's lower bound on s given by (2.57) and simplified in (2.60). Note that this lower bound on the *maximum* dimension is equal to Proposition 2.4's lower bound on the *minimum* dimension. For example, the (24,12,8) extended Golay code has $\Delta = \Delta^{\perp} = 5$, with (2.60) indicating that the maximum dimension is $s \geq 2$, while (2.62) indicates that the minimum dimension in the variable dimension region is $s' \geq 2$.

We now return to the upper bound $s \leq \min(k, n-k)$ and note that a systematic ordering of symbol positions will result in dimensions meeting this bound. Using the definitions of Δ and $\Delta^{\perp}$ this is equivalent to

$$s = \min\left(d_{min} - 1 + \Delta, d^{\perp}_{min} - 1 + \Delta^{\perp}\right) \qquad \text{(systematic form)}. \tag{2.63}$$

Combining this upper bound on s with Proposition 2.4 we have the following upper and lower bounds on the trellis dimensions in the variable dimension region,

$$\max\left[0, \min\left(d_{min}-1-\Delta, d^{\perp}_{min}-1-\Delta^{\perp}\right)\right] \leq s_l \leq \min\left(d_{min}-1+\Delta, d^{\perp}_{min}-1+\Delta^{\perp}\right), \quad d \leq l \leq n-d. \tag{2.64}$$

For example, consider a code and its dual with smallest minimum distance d and corresponding Δ, denoted Δ_d, and assume that $\Delta_d < d - 1$. Then from (2.64) the range in trellis dimensions in the

variable dimension region will be $\pm\Delta_d$, with respect to fixed trellis dimension of $d-1$ at depths $d-1$ from either end of the trellis. This variability is represented in Figure 2.9, which shows the bounding envelope of the trellis dimensions for all symbol position permutations.

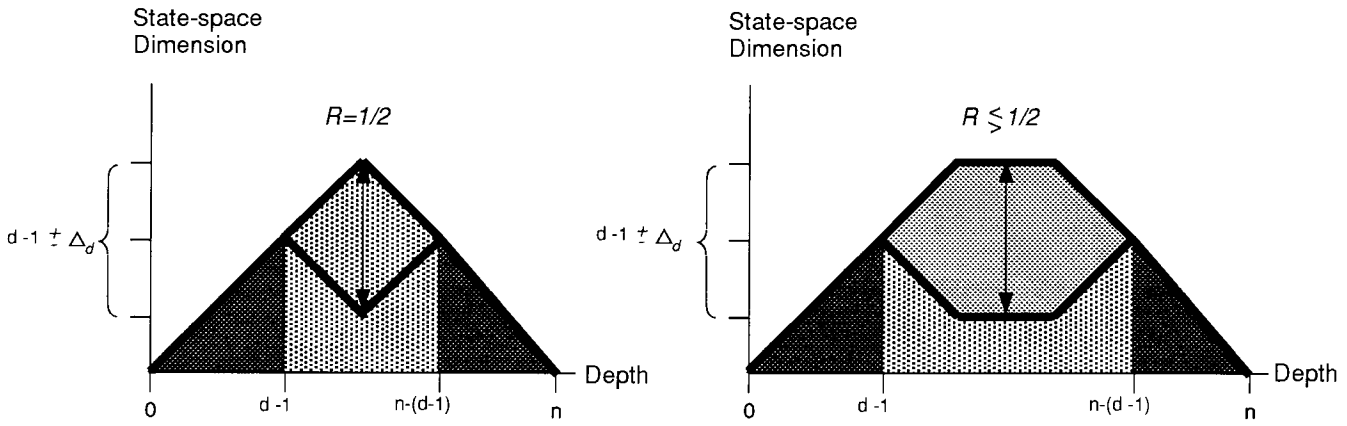


Figure 2.9 Bounding Envelopes of the Trellis Dimensions
The heavy lines indicate upper and lower bounds on the trellis dimensions. Near either end of the trellis, the dark shading indicates the *fixed dimension regions*. The light shading indicates the *variable dimension region*. Examples shown are for $\Delta_d < d-1$.

The bounds indicate that a substantial decrease in trellis dimension is achievable only for codes that have a large Δ_d, or equivalently a low minimum distance d relative to the corresponding Singleton bound, $d_{min} \leq n-k+1$ or $d_{min}^{\perp} \leq k+1$. For example, codes with low d, such as binary BCH codes for increasing n, may have a maximum trellis dimension that is considerably smaller than $n-k$. Indeed, in [23], Kasami *et al* have recently found such trellises for some binary BCH codes. However, for codes with d relatively close to the Singelton bound, (2.64) shows that a large decrease in state-space dimension is not possible. These results, in addition to those of

Muder, strengthen support for the conjecture [17] that when d_{min} is maximized for a given n and k, then s is close to its maximum possible value.

2.5.1 Discussion

Beginning with Bahl *et al* [15] and Wolf [8] the advantage of trellis decoding over brute force correlation decoding was demonstrated for high rate codes, using the q^{n-k} upper bound on the number of states. In [15] [9] [22] and elsewhere, the fact that the trellis dimensions are affected by symbol position permutations was observed. In [9] it was emphasized that this property should be exploited to simplify decoding. However, as discussed in [9] and [22], the variability of the trellis dimensions for general codes was not explicitly known. From Muder's lower bounds on the maximum trellis dimension and from the improved and new bounds developed here, the dependence of the trellis dimensions on the specific code has been made more explicit. Lower bounds on the maximum trellis dimension and on the minimum trellis dimension (in the variable dimension region) are given in terms of the code parameters $n, k, d_{min}, d_{min}^{\perp}$. In particular, the lower bound on the minimum trellis dimension appears to be the first such bound developed. This bound can be used to demonstrate that some codes will not have a small number of states relative to the worst case of $\min\left(q^k, q^{n-k}\right)$. For example, the trellis dimensions of MDS codes are completely unaffected by symbol position permutation, and remain at their worst case values. The bounds indicate that only codes (and their duals) that have a smallest minimum distance $d = \min\left(d_{min}, d_{min}^{\perp}\right)$ significantly less than the corresponding Singleton bound can possibly have a small number of states relative to the worst case of $\min\left(q^k, q^{n-k}\right)$.

Chapter 3

Trellis Decoding with Pruning

What defeats us ... is the "curse of dimensionality".

R. Bellman [24]

MAXIMUM–LIKELIHOOD decoding using the Viterbi algorithm rapidly becomes impractical as the trellis dimensions increase. This exponential growth of the number of trellis states is an example of Bellman's "curse of dimensionality" [24] [25] in dynamic programming. With larger trellises, we are thus motivated to find strategies to save decoding effort.

As discussed in Chapter 1, the trellis search can be reduced by judiciously discarding paths based on soft-decision metrics; this avoids an increase in error probability due to coarse quantization or hard-decision decoding which effectively discard likelihood information at the outset for *all* paths. However, some increase in error probability will be incurred if paths are discarded from the search without assurance that they are not the ML path. We will refer to such action as pruning. While

decoding with pruning will result in an increased probability of error relative to ML decoding, this increase will be tolerable (or perhaps negligible) when the pruning decoder is well designed.

Two aspects of decoding with pruning are discussed in this chapter. The first is the choice and use of an appropriate metric for pruning on a tree or trellis. The second is the task of *contender sifting*. Contender sifting refers to the selection of some number of the best metrics from among a larger number of competing metrics. A method for contender sifting is introduced that is considerably more efficient than schemes based on sorting or selection.

In Section 3.1 we briefly review the main types of tree and trellis decoding algorithms. In Section 3.2 we describe partial trellis searches, and define some terminology for use in later sections.

In Section 3.3 we find an appropriate metric for pruning on a tree or trellis. The approach taken is somewhat different from [26] in its description of the problem and in its modeling of the decoder's knowledge of branch-likelihoods in the unexplored part of the tree or trellis. It is shown that the decoder should discard heads in the obvious manner (by first exploiting merges in the trellis and then pruning the 'worst' heads first as indicated by the metric) and that in some circumstances the metric should be altered from that usually used. Such situations include unequal *a priori* information-symbol probabilities, non-linear codes, and non-symmetric, non–binary discrete-memoryless-channels (DMCs).

In Section 3.4 we discuss some previous approaches to contender sifting and present a new contender sifting method. The new method can sift out a sorted set of the best M of $2M$ metrics using a worst case number of comparisons that is linear in M. This might seem surprising, since it is well known that comparison-based sorting of n items requires $O(n \log_2 n)$ comparisons [27]. The efficiency of the contender sifting method is attained by exploiting an ordering that is inherent

in the metrics. Specific applications to the decoding of binary linear block codes and binary linear convolutional codes are presented.

3.1 Tree/Trellis Decoding Algorithms

Full searches of a code tree consider all paths through the tree and then select the ML path. The VA was introduced for decoding convolutional codes [12] and it exploits the trellis structure [11]. Its decoding efficiency arises from the observation that of any heads that share the same tail(s), only the single best of these heads needs be considered further into the trellis. The VA extends heads to the next depth of the trellis, and then retains only the single best head for each state. In this way, the VA discards any path as soon as it has proven to be inferior to some other path. This yields the same decoder output as if *all* paths through the trellis were considered. The VA has also been used in several other applications [11][28] besides decoding of convolutional codes, including the decoding of linear block codes since they may be represented by a trellis [15][8][9][16].

Several other search algorithms have been used that may partially explore the tree or trellis. They are usually classified as breadth-first, depth-first, and metric-first type searches [29].

The VA and the M algorithm [30][31] are examples of breadth-first searches. M algorithm decoding on a trellis proceeds just as the VA at each depth, except that only the best M states are retained instead of the full trellis width.[17] This pruning used by the M algorithm will result in an increased probability of error relative to the VA. Some other breadth-first algorithms are similar to the M algorithm except that the number of heads retained is dynamic; a head is retained until its metric differs from the current 'best' head by a certain amount [32]. This approach can lower the average computational effort at the expense of a variable decoding delay.

[17] For use with very long sequences of convolutionally encoded data, the path storage length is truncated to L symbols, and the algorithm is then referred to as the (M,L) algorithm.

An example of a depth-first search is the Fano algorithm [33] which pursues a single head into the tree until its metric falls below an adaptive threshold. When a candidate head fails in this manner, the algorithm backtracks into the tree to follow another head. The Fano algorithm thus makes a variable number of attempts at following an acceptable head.

The third class of decoding algorithms are the metric-first approaches, or *stack* decoding algorithms. Stack decoding algorithms maintain lists of candidate heads of various lengths ranked according to their metrics. In the Zigangirov-Jelinek stack algorithm [34][35] the search is carried out by extending the highest ranked head to form new heads, then merging these metrics with those already ranked. There are several variants of the stack algorithms [29], which in general attempt to reduce the variability of the decoding effort and the probability of erasing the correct path.

3.2 A Description of Partial Trellis Searches

Pruning decoders begin operation at the root node of the trellis, with no heads as yet explored. The search begins by extending heads from the root node and computing their metrics. Using these heads and metrics the decoder then decides which heads to retain for possible further consideration, and which heads to discard. The heads that are compared are referred to as *contenders*. Those contenders that are retained are referred to as *survivors*. The action of selecting the contenders to be retained is referred to as *contender sifting*. The discarded heads will be referred to either as *defeated* or *pruned*, and the distinction between these is as follows. A *defeated* head is one that is discarded from further consideration due to the fact that a test of its metric and state is sufficient to prove that it will have a poorer *path* metric relative to at least one other path. A *pruned* head is one that is also discarded from further consideration, but the test of its metric and state used to decide to discard it is insufficient to prove that it will have a poorer *path* metric relative to at least one other path.

The decoding search is performed in a sequence of *stages*, as shown in Figures 3.1 and 3.2. Each stage consists of the following steps:

1. Extension of some or all of the previous stage's survivors, to form contenders.
2. Updating of the metrics for the contenders.
3. Determination of the heads to be defeated.
4. Determination of the heads to be pruned.

It will be useful to view the decoder as acting on sets of heads or on corresponding sets of codewords, as described below.

Since each head corresponds to one or more codewords, we may view the decoder as acting on the set of all q^k codewords, as shown in Figure 3.1. At the input to the first stage, all q^k codewords are contenders, and by the end of the stage some may have been defeated and some others may have been pruned. The survivors of stage 1 become the contenders for stage 2, and the process is repeated. Thus, the decoder progressively reduces the number of survivors stage-by-stage, with a single survivor at the end of the final stage being the decoder's output.

Figure 3.2 represents the decoder as acting on sets of heads, instead of acting on sets of codewords as in Figure 3.1. Some or all of the survivors of a previous stage are extended to form contenders. These contenders are then considered by the decoder for possible elimination.

In Figure 3.1, the set of contender codewords for stage m is denoted S_m. S_{D_m} and S_{P_m} denote the sets of defeated and pruned codewords, respectively, at stage m. These sets of codewords are represented by corresponding sets of heads in Figure 3.2, and are denoted S^h_m, $S^h_{D_m}$, and $S^h_{P_m}$, respectively.

The breadth-first, depth-first, and metric-first decoding algorithms discussed in §3.1 can be described by this model of partial trellis searches as follows. In a breadth-first decoder, each stage begins with extending all survivors, typically one branch further into the trellis, to form contenders of equal length. Contenders may be defeated and pruned at each stage, and there are $m_{\text{final}} = n$ stages in total. In contrast, the depth-first and metric-first algorithms do not extend all survivors at each stage. For example, the stack algorithm or the Fano algorithm extend only a single survivor per stage, and the total number of stages m_{final} is a random variable. We can view such backtracking searches as not pruning at any stage except the final stage.

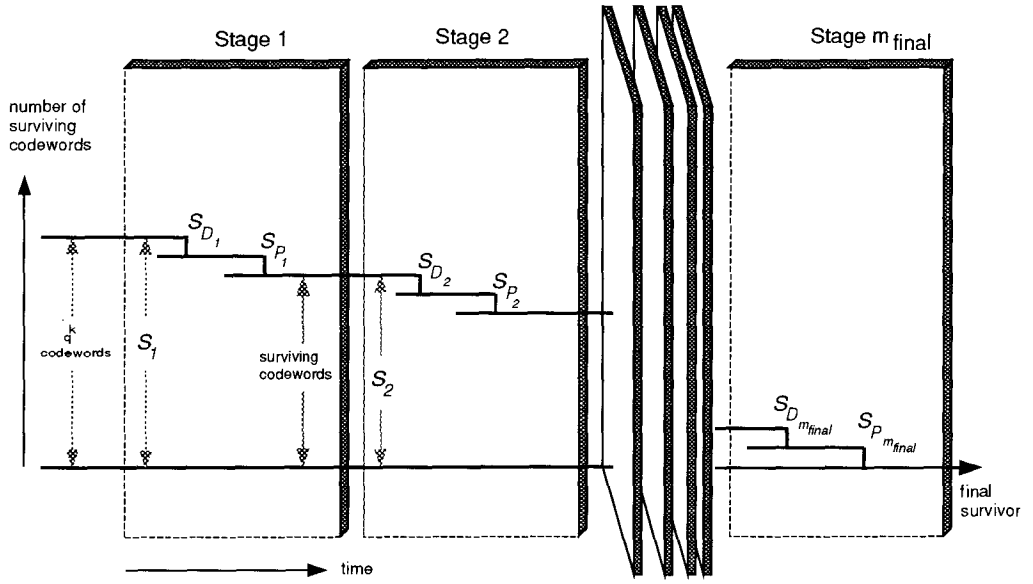


Figure 3.1. Representation of Pruning Decoder Operation as Acting on Sets of Codewords
The decoder begins with the set of all q^k codewords and progressively reduces the number of surviving codewords. S_m is the set of contender codewords for stage m. S_{Dm} and S_{Pm} are sets of defeated and pruned codewords, respectively, at stage m.

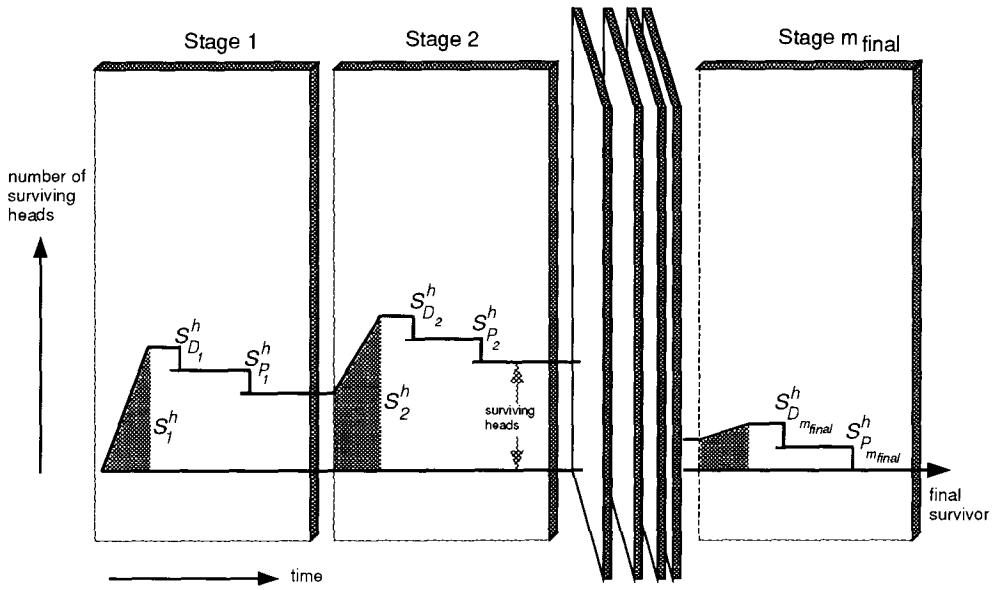


Figure 3.2. Representation of Pruning Decoder Operation as Acting on Sets of Heads
The shaded region within each stage m represents the formation of the contenders S^h_m from the previous stage's survivors. The defeated heads $S^h_{D_m}$ and the pruned heads $S^h_{P_m}$ are chosen from the contenders S^h_m. The set $S^h_{D_m}$ consists of all heads whose descendants are the set of codewords S_{D_m} in Figure 3.1. Similarly, $S^h_{P_m}$ corresponds to the set of pruned codewords S_{P_m} in Figure 3.1.

3.3 Decoders Constrained by Pruning

3.3.1 A General Decoding Metric and its use in Trellis Decoding

In this section we assume that we are *given* some survivor extension method; i.e. we are given the decoder's method for extending the set of survivors from the previous stage. We find a decoding metric and show how it should be used in trellis decoding in order to minimize the probability of error, given the fact that the decoder is constrained to use pruning. We do not completely specify how to carry out the pruning in terms of, for example, the number of heads to prune; that and the choice of the survivor extension method can be separated from the choice of an appropriate metric.

The discussion in this section is divided into five steps labeled (i)-(v), with the following organization. Step (i) expresses the probability of error in terms of discarding codewords from a list of all codewords, as illustrated in Figure 3.1. Step (ii) incorporates into the expression for the probability of error the fact that multiple codewords are discarded per head, as illustrated in Figure 3.2. Step (iii) discusses the constraint on the minimization of the probability of error imposed by the fact that the codeword likelihood information is incomplete (since only part of the trellis has been explored). We use a model of the decoder's knowledge of the branch–likelihoods on the unexplored tails that specializes Massey's 'random-tail' approach [26]. Step (iv) discusses the constraint that the pruning of heads is carried out stage-by-stage. Finally, Step (v) finds an equivalent form of the decoding metric that involves only quantities computed over the explored portion of the trellis.

(i) Probability of Error

Since the action of the decoder is to discard codewords in multiple stages, as shown in Figure 3.1, it is natural to express the probability of error as the sum of contributions from each stage.

First, the probability of correct decoding is

$$Pr(\text{correct }) = \int_{\mathbf{R}} Pr(\text{correct } \mid \mathbf{r}) f(\mathbf{r})\, d\mathbf{r} \tag{3.1}$$

where $Pr(\text{ correct } \mid \mathbf{r}) = Pr(\mathbf{c}_{\text{out}} = \mathbf{c}_{\text{transmitted}} \mid \mathbf{r})$, with $\mathbf{c}_{\text{out}}$ being the codeword released by the decoder. We also use the notation $Pr(\mathbf{c} \mid \mathbf{r})$ to denote the probability that the codeword $\mathbf{c}$ was transmitted given that $\mathbf{r}$ was received. Now, using (3.1) and the fact that $\sum_{\mathbf{c}\in\mathbf{C}} Pr(\mathbf{c} \mid \mathbf{r}) = 1$ we may write

$$\begin{aligned} Pr(\text{error}) &= 1 - Pr(\text{ correct}) \\ &= \int_{\mathbf{R}} f(\mathbf{r})\, d\mathbf{r} - \int_{\mathbf{R}} Pr(\mathbf{c}_{\text{out}} \mid \mathbf{r}) f(\mathbf{r})\, d\mathbf{r} \\ &= \int_{\mathbf{R}} \sum_{\mathbf{c}\in\mathbf{C}} Pr(\mathbf{c} \mid \mathbf{r})\, f(\mathbf{r})\, d\mathbf{r} - \int_{\mathbf{R}} Pr(\mathbf{c}_{\text{out}} \mid \mathbf{r}) f(\mathbf{r})\, d\mathbf{r} \\ &= \int_{\mathbf{R}} \sum_{\substack{\mathbf{c}\neq\mathbf{c}_{\text{out}} \\ \mathbf{c}\in\mathbf{C}}} Pr(\mathbf{c} \mid \mathbf{r})\, f(\mathbf{r})\, d\mathbf{r}\,. \end{aligned} \tag{3.2}$$

Equation (3.2) expresses the probability of error as the expectation over all possible demodulator outputs $\mathbf{r} \in \mathbf{R}$ of the probability that any unchosen codeword given $\mathbf{r}$ is correct.

All of the codewords appearing in the sum in (3.2) are by definition either *defeated* or *pruned.* Hence we may split the sum in (3.2) to give

$$Pr(\text{error}) = \int_{\mathbf{R}} \left[\sum_{\mathbf{c}\in S_D} Pr(\mathbf{c} \mid \mathbf{r}) + \sum_{\mathbf{c}\in S_P} Pr(\mathbf{c} \mid \mathbf{r}) \right] f(\mathbf{r})\, d\mathbf{r} \tag{3.3}$$

The integral of the terms in the first sum in (3.3) is the contribution to the error probability from decoding trials in which a defeated codeword was in fact correct. Similarly, the integral of the terms in the second sum in (3.3) is the contribution to the error probability from decoding trials in which a pruned codeword was correct.

To minimize (3.3), it is sufficient to minimize its integrand for each $\mathbf{r}$, which can be written as

$$\sum_{\mathbf{c}\in\mathbf{C}} [I(\mathbf{c} \in S_D) + I(\mathbf{c} \in S_P)]\, Pr(\mathbf{c} \mid \mathbf{r})\, f(\mathbf{r}) \tag{3.4}$$

where $I(\cdot)$ is unity if its argument is true and is zero otherwise. As well, since the decoder discards disjoint sets of codewords at each stage we can rewrite (3.4) as

$$\sum_{m=1}^{m_{\text{final}}} \sum_{\mathbf{c}\in\mathbf{C}} [I(\mathbf{c} \in S_{D_m}) + I(\mathbf{c} \in S_{P_m})]\, Pr(\mathbf{c} \mid \mathbf{r})\, f(\mathbf{r}) \ . \tag{3.5}$$

(ii) Multiple Codewords Discarded per Head

When the decoder discards a head it consequently discards all of the codewords that descend from the head, so that the decoder is not free to choose individual codewords to discard in minimizing (3.5)[18]. Hence, the set of discarded codewords $S_{P_m} + S_{D_m}$ in (3.5) consists of all descendant codewords of the pruned heads $S^h_{P_m}$ and of the defeated heads $S^h_{D_m}$. To incorporate this fact into (3.5) we first note that for any codeword,

$$\begin{aligned} Pr(\mathbf{c} \mid \mathbf{r}) f(\mathbf{r}) &= f(\mathbf{r} \mid \mathbf{c}) Pr(\mathbf{c}) \\ &= f\Big(\mathbf{r}^h, \mathbf{r}^t \mid \mathbf{c}^h, \mathbf{c}^t\Big) Pr\Big(\mathbf{c}^h, \mathbf{c}^t\Big) \\ &= f\Big(\mathbf{r}^h \mid \mathbf{c}^h, \mathbf{c}^t\Big) f\Big(\mathbf{r}^t \mid \mathbf{r}^h, \mathbf{c}^h, \mathbf{c}^t\Big) Pr\Big(\mathbf{c}^h\Big) Pr\Big(\mathbf{c}^t \mid \mathbf{c}^h\Big) \\ &= f\Big(\mathbf{r}^h \mid \mathbf{c}^h\Big) f(\mathbf{r}^t \mid \mathbf{c}^t) Pr\Big(\mathbf{c}^h\Big) Pr\Big(\mathbf{c}^t \mid \mathbf{c}^h\Big) \end{aligned} \tag{3.6}$$

where in the final step we have used the fact that the channel is assumed to be memoryless.[19] Using (3.6) in (3.5), and since the discarded codewords at stage m are chosen from the descendants

[18] See also Figure 3.2.

[19] Also, in (3.6), $Pr(\mathbf{c}^h) = Pr(c_1)Pr(c_2 \mid c_1)Pr(c_3 \mid c_2, c_1) \cdots Pr(c_l \mid c_{l-1}, \ldots, c_2, c_1)$ and $Pr(\mathbf{c}^t \mid \mathbf{c}^h) = Pr(c_{l+1} \mid c_l, \ldots, c_2, c_1)Pr(c_{l+2} \mid c_{l+1}, \ldots, c_2, c_1) \cdots Pr(c_n \mid c_{n-1}, \ldots, c_2, c_1)$.

of S_m^h, we can replace $\sum_{\mathbf{c}\in\mathbf{C}}$ in (3.5) by $\sum_{\mathbf{c}^h\in S_m^h} \sum_{\forall \mathbf{c}_t \leftrightarrow \mathbf{c}_h}$, where the last sum generates all tails $\mathbf{c}^t$ of a head $\mathbf{c}^h$, to obtain

$$\sum_{m=1}^{m_{\text{final}}} \sum_{\mathbf{c}^h\in S_m^h} \left[I\left(\mathbf{c}^h \in S_{P_m}^h\right) + I\left(\mathbf{c}^h \in S_{D_m}^h\right)\right] f\left(\mathbf{r}^h \mid \mathbf{c}^h\right) Pr\left(\mathbf{c}^h\right) \sum_{\forall \mathbf{c}^t \leftrightarrow \mathbf{c}^h} f(\mathbf{r}^t \mid \mathbf{c}^t) Pr\left(\mathbf{c}^t \mid \mathbf{c}^h\right) . \tag{3.7}$$

Equation (3.7) makes explicit the fact that the discard decisions are made at the heads and that *all* descendant codewords of each head are consequently discarded.

(iii) Optimization Constraint: Incomplete Codeword Likelihood Information

Minimization of (3.7) depends on what codeword likelihood information is available; some of the likelihood information will not be available due to the fact that the search is incomplete. The codeword likelihood information available at any stage can be considered to be split into two portions, with the first portion consisting of branch-likelihoods on the explored head, and the second portion consisting of less specific branch-likelihood information available on the unexplored tail. To clarify this let us first consider (3.6), and rewrite it as a product of terms that correspond to the head and tail, viz

$$\underbrace{\left[f\left(\mathbf{r}^h \mid \mathbf{c}^h\right) Pr\left(\mathbf{c}^h\right)\right]}_{g_1(\mathbf{r}^h,\mathbf{c}^h)} \underbrace{\left[f(\mathbf{r}^t \mid \mathbf{c}^t) Pr\left(\mathbf{c}^t \mid \mathbf{c}^h\right)\right]}_{g_2(\mathbf{r}^t,\mathbf{c}^t,\mathbf{c}^h)} . \tag{3.8}$$

We have labeled the first term in square brackets in (3.8) as a function $g_1\left(\mathbf{r}^h, \mathbf{c}^h\right)$, and have labeled the second term as a function $g_2\left(\mathbf{r}^t, \mathbf{c}^t, \mathbf{c}^h\right)$.

For an explored head $\mathbf{c}^h$ the decoder has computed $f\left(\mathbf{r}^h \mid \mathbf{c}^h\right) Pr\left(\mathbf{c}^h\right)$. In other words, for an explored head $\mathbf{c}^h$ the decoder has available the function $g_1\left(\mathbf{r}^h, \mathbf{c}^h\right)$ precisely.

For an unexplored tail $\mathbf{c}^t$ the decoder has not computed $f\left(\mathbf{r}^t \mid \mathbf{c}^t\right) Pr\left(\mathbf{c}^t \mid \mathbf{c}^h\right)$. In other words, the decoder does not have available $g_2\left(\mathbf{r}^t, \mathbf{c}^t, \mathbf{c}^h\right)$. We need to consider how the decoder should

act given that it doesn't have the function $g_2(\mathbf{r}^t, \mathbf{c}^t, \mathbf{c}^h)$ available precisely. We consider two cases. The first case is where the head $\mathbf{c}^h$ can be classified as defeated. In this case the competing heads at a node share all their tails, so the decoder can defeat $\mathbf{c}^h$ without needing to know $g_2(\mathbf{r}^t, \mathbf{c}^t, \mathbf{c}^h)$. This is precisely the basis used for discarding codewords in the Viterbi algorithm. The second case is where the head $\mathbf{c}^h$ cannot be classified as defeated. In this case the decoder is constrained to *estimate* the function $g_2(\mathbf{r}^t, \mathbf{c}^t, \mathbf{c}^h)$.

The estimation of the function $g_2(\mathbf{r}^t, \mathbf{c}^t, \mathbf{c}^h)$ depends on what information is available to the decoder with regard to the unexplored tail. For this we have to assume (and justify) a decoder model. In other words, we need to state what *a priori* and *a posteriori* information regarding an unexplored tail should be assumed available to the decoder. We also need to state how the decoder may be constrained in utilizing this information.

We will use a model of sequential decoding that is essentially that used in [26]. The decoder is considered to have no specific knowledge of the code symbol on an unexplored branch, but as a head is extended to explore the branch the decoder gains knowledge of the code symbol. (This code symbol is then used to look up the appropriate symbol metric for use in updating the head metric).

To consider how the decoder will estimate $g_2(\mathbf{r}^t, \mathbf{c}^t, \mathbf{c}^h) = f(\mathbf{r}^t \mid \mathbf{c}^t)\, Pr(\mathbf{c}^t \mid \mathbf{c}^h)$ we first recall that

$$f(\mathbf{r}^t \mid \mathbf{c}^t) = \prod_{i=l+1}^{n} f(r_i \mid c_i). \tag{3.9}$$

We assume that the decoder has computed $(f(r_i \mid 0), f(r_i \mid 1), \ldots, f(r_i \mid q-1))$ for every symbol position i. In other words, we assume that the decoder has computed the symbol metrics for every symbol position. However, on an unexplored branch the decoder has (as yet) effectively no specific information as to which likelihood to use. This corresponds to an uncertainty as to which likelihood will be assigned to an unexplored branch. This uncertainty can be quantified and used to estimate

$f(\mathbf{r}^t \mid \mathbf{c}^t)$ as follows. Let us assume that the decoder has, in advance, utilized knowledge of the code and the *a priori* information-symbol probabilities, to compute the probabilities $\{Q_i(j)\}_{\substack{j=0,1,\ldots,q-1 \\ i=1,2,\ldots,n}}$ of the occurrence of the channel-symbol j for position i. (For example, it is common to use linear codes with equiprobable information-symbols, in which case $Q_i(j) = 1/q$ for $j = 0, 1, \ldots, q-1$ and $i = 1, 2, \ldots, n$.) On an unexplored branch at position i, the decoder is then taken to be constrained to model symbol j as occurring according to the probability assignment $Q_i(j)$[20]. Consequently, the decoder will not be able to distinguish between different unexplored tails of the same length, since the information available (namely, $\{Q_i(j)\}_{\substack{j=0,1,\ldots,q-1 \\ i=l+1,\ldots,n}}$ and $\{ f(r_i \mid j) \}_{\substack{j=0,1,\ldots,q-1 \\ i=l+1,\ldots,n}}$) is the same for any of them. Hence the decoder's estimate of $f(\mathbf{r}^t \mid \mathbf{c}^t)$ is actually only a function of $\mathbf{r}^t$ (and the probability assignment $Q_i(j)$). Accordingly, we let $f_0(\mathbf{r}^t)$ denote the decoder's estimate of $f(\mathbf{r}^t \mid \mathbf{c}^t)$.

We now show that as a consequence of this model, $f_0(\mathbf{r}^t)$ can be interpreted and computed in a simple way. Consider the pdf of $\mathbf{r}^t$, which is given by

$$f(\mathbf{r}^t) = \sum_{\mathbf{c}^t} f(\mathbf{r}^t \mid \mathbf{c}^t) Pr(\mathbf{c}^t) \tag{3.10}$$

where the sum includes every tail $\mathbf{c}^t$ of length $n - l$. Using $f_0(\mathbf{r}^t)$ as the decoder's estimate of $f(\mathbf{r}^t \mid \mathbf{c}^t)$ in (3.10) we obtain an approximation to $f(\mathbf{r}^t)$ as

$$\sum_{\mathbf{c}^t} f_0(\mathbf{r}^t) Pr(\mathbf{c}^t) = f_0(\mathbf{r}^t). \tag{3.11}$$

This shows that $f_0(\mathbf{r}^t)$ can be interpreted as the approximate pdf of $\mathbf{r}^t$. Accordingly, we compute $f_0(\mathbf{r}^t)$ as

$$f_0(\mathbf{r}^t) = \prod_{i=l+1}^{n} \sum_{j=1}^{q} f(r_i \mid j) Q_i(j). \tag{3.12}$$

[20] This is essentially a specialization of the 'random-tail' approach used in [26]. We will later discuss the significance of using $Q_i(j)$ versus other distributions.

The contribution to (3.7) if a head is discarded is

$$f\Big(\mathbf{r}^h \mid \mathbf{c}^h\Big) Pr\Big(\mathbf{c}^h\Big) \sum_{\forall \mathbf{c}^t \leftrightarrow \mathbf{c}^h} f(\mathbf{r}^t \mid \mathbf{c}^t) Pr\Big(\mathbf{c}^t \mid \mathbf{c}^h\Big). \tag{3.13}$$

As discussed above, the decoder has computed $f(\mathbf{r}^h \mid \mathbf{c}^h) Pr(\mathbf{c}^h)$, and can estimate $f(\mathbf{r}^t \mid \mathbf{c}^t)$ as $f_0(\mathbf{r}^t)$. Using this in (3.13) we obtain

$$f\Big(\mathbf{r}^h \mid \mathbf{c}^h\Big) Pr\Big(\mathbf{c}^h\Big) \sum_{\forall \mathbf{c}^t \leftrightarrow \mathbf{c}^h} f_o(\mathbf{r}^t) Pr\Big(\mathbf{c}^t \mid \mathbf{c}^h\Big) \;=\; f\Big(\mathbf{r}^h \mid \mathbf{c}^h\Big)\, Pr\Big(\mathbf{c}^h\Big)\, f_o(\mathbf{r}^t) \tag{3.14}$$

as the expected contribution to the error probability if a head is pruned.

For later convenience, we choose to write

$$f_0(\mathbf{r}^t) = \prod_{i=l+1}^{n} f_0(r_i) \tag{3.15}$$

with each term in the product defined as

$$f_0(r_i) = \sum_{j=1}^{q} f(r_i \mid j)\, Q_i(j). \tag{3.16}$$

(iv) Optimization Constraint: Stage by Stage Decoding

Here we consider that the decoder must discard heads stage by stage, as illustrated in Figure 3.2. The decoder is *constrained* to decide at each stage on which heads to discard, i.e. the decoder is constrained to consider each stage independently of future stages. This corresponds to the decoder not being able to predict which heads will be discarded in future stages. In considering which heads to discard from S_m^h, the decoder should first discard those heads that can be classified as defeated, since (as discussed earlier) a defeated head's descendants will contribute less to (3.7) than

its competitor's descendants. Next, in considering which heads to discard from those remaining (i.e. $S_m^h - S_{D_m}^h$) the decoder can minimize its expected contribution to (3.7) by using (3.14) as the expected contribution for pruning a head. In summary, the decoder minimizes

$$\sum_{\mathbf{c}^h \in S_{P_m}^h} f\left(\mathbf{r}^h \mid \mathbf{c}^h\right) Pr\left(\mathbf{c}^h\right) f_o\left(\mathbf{r}^t\right) \tag{3.17}$$

at each successive stage m, where $S_{P_m}^h \subset S_m^h - S_{D_m}^h$.

(v) An Equivalent Decoding Metric

The above results indicate that the heads to be pruned at each stage are those whose pdf $f\left(\mathbf{r}^h \mid \mathbf{c}^h\right) Pr\left(\mathbf{c}^h\right)$ *weighted* by $f_o\left(\mathbf{r}^t\right)$ are lowest. The effect of the weighting by $f_o\left(\mathbf{r}^t\right)$ is that if this 'expected tail' is quite likely, then further consideration of the head is favoured, since the unexplored branches occur frequently and hence may contribute significantly to the error probability. On the other hand if this 'expected tail' is unlikely, then if the head is also unlikely then discarding the head will not significantly contribute to the error probability.

Instead of minimizing (3.17), we may equivalently minimize (3.17) divided by any positive quantity that is independent of each $\mathbf{c}^h$. Suppose we choose to divide (3.17) by $f_0\left(\mathbf{r}^h\right) f_0\left(\mathbf{r}^t\right)$, where $f_0\left(\mathbf{r}^h\right)$ is defined similarly as done for $f_o\left(\mathbf{r}^t\right)$ in (3.15), viz

$$\begin{aligned} f_0\left(\mathbf{r}^h\right) &= \prod_{i=1}^{l} f_0(r_i) \\ &= \prod_{i=1}^{l} \sum_{j=1}^{q} l_{ij}\, Q_i(j)\,. \end{aligned} \tag{3.18}$$

The decoder can now aim to minimize

$$\sum_{\mathbf{c}^h \in S_{P_m}^h} \frac{f\left(\mathbf{r}^h \mid \mathbf{c}^h\right) Pr\left(\mathbf{c}^h\right)}{f_0\left(\mathbf{r}^h\right)} \tag{3.19}$$

at each successive stage m, where $S^h_{P_m} \subset S^h_m - S^h_{D_m}$. From (3.19) the metric is now

$$\frac{f(\mathbf{r}^h \mid \mathbf{c}^h) Pr(\mathbf{c}^h)}{f_o(\mathbf{r}^h)} \quad . \tag{3.20}$$

The advantage of (3.19) over (3.17) is that it only involves quantities computed over the explored portion of the trellis. Each head likelihood weighting has been normalized by $f_0(\mathbf{r}^h) f_0(\mathbf{r}^t)$, which results in a weighting term involving only information available on the explored branches. Consequently, the decoder fits the usual definition of *sequential decoding*, i.e. decoding that sequentially examines branches of the trellis so that at any stage the decoder's choice of which heads to discard and to extend do not depend upon received branches deeper in the trellis. This definition of sequential decoding is equivalent to that of Jacobs and Berlekamp [36], except for our explicit allowance of trellis decoding in addition to tree decoding. The decoder is sequential in that the computation of its metric does not depend on received branches deeper in the trellis, although we have utilized some knowledge of the unexplored branch likelihoods in the metric derivation.

3.3.2 Discussion

Eqn. (3.19) simply states that the decoder should, during each stage, first update the contender metrics using (3.20), then discard those contenders that can be *defeated*, and then prune only the poorest of the remaining contenders. In other words, the decoder should discard contenders in a rather obvious manner, by first exploiting merges in the trellis and then by pruning contenders 'worst-first'. However, the metric (3.20) differs slightly from that used previously, as we next describe.

In [26] and the related discussion in [37], sequential decoding is modeled as being equivalent to the decoding of variable length codes that have been appended with randomly coded tails. The

sequential decoder's knowledge of the unexplored codeword tails is thus modeled as if randomly chosen code symbols were used. In modeling the decoder's knowledge of the unexplored codeword tails, our approach specializes the use of random code tails as in [26][37], by assuming that the decoder has precomputed the channel-symbol probabilities that are determined by the code and information-symbol distribution used. This results in some minor differences between our "normalizing" term $f_0(r_i)$ given by (3.16) and the corresponding term denoted $P_0(y_i)$ in [26][37]. Specifically, from [26][37, pg 45],

$$P_o(y_i) = \sum_j Pr(y_i \mid j) Q_{\text{random}}(j) \tag{3.21}$$

where y_i is a DMC output and where $Q_{\text{random}}(j)$ is a random-coding probability assignment for a channel-symbol to take on the value j. From (3.16)

$$f_o(r_i) = \sum_j f(r_i \mid j)\, Q_i(j) \tag{3.22}$$

where $Q_i(j)$ is the channel-symbol probability distribution for channel symbols at depth i taking on the value j, for the *specific code and the specific information-symbol distribution used.* The two distributions, $Q_{\text{random}}(j)$ and $Q_i(j)$, differ in two minor respects. First, $Q_{\text{random}}(j)$ is not a function of the symbol position, while $Q_i(j)$ accommodates differing probability assignments for different positions. An example of where the two distributions will differ in this regard is the case of unequal information-symbol probabilities. Second, $Q_{\text{random}}(j)$ is a 'random-coding' probability distribution in the sense that it is usually taken [37, pg 45][2] to be the maximizing input-distribution in the computation of R_0 [3]. $Q_{\text{random}}(j)$ will then be determined only by the channel. This differs from $Q_i(j)$ which is determined, not by the channel, but by the specific code and information-symbol distribution used. However, for the typical case of a symmetric DMC [2] (or a binary-input DMC [37]), we will have $Q_{\text{random}}(j) = 1/q, j = 0, 1, \ldots, q-1$, which will equal $Q_i(j)$ for the (also typical) case of linear codes with equiprobable codewords. Atypical decoding

situations that involve non-linear codes, or non-symmetric, non-binary-input DMCs, or unequal *a priori* information-symbol probabilities, can result in differences between these two distributions. For most coding applications the metric (3.20) is identical to that used in [26][37].

3.3.3 Breadth-First Decoding

Here we consider breadth-first decoding with equiprobable codewords, for which the metric (3.20) can be simplified and put into a more convenient form. This breadth-first metric will be used extensively in the next chapter.

In breadth-first decoding, heads are compared for pruning when they are of equal length. In minimizing (3.19) the decoder can then equivalently minimize

$$\sum_{\mathbf{c}^h \in S^h_{Pm}} f\left(\mathbf{r}^h \mid \mathbf{c}^h\right) \tag{3.23}$$

at each stage, since for equiprobable codewords and equal length heads we can ignore the common terms $Pr(\mathbf{c}^h)$ and $f_0(\mathbf{r}^h)$.

To choose heads that will minimize (3.23), we can also compare heads in the following manner. First, a head metric $f(\mathbf{r}^h \mid \mathbf{c}^h)$ can be compared to other head metrics by favouring to prune first those heads that have the largest $1/f(\mathbf{r}^h \mid \mathbf{c}^h)$. As well, we can multiply this by any positive factor that is independent of $\mathbf{c}^h$. If we choose to multiply by $f(\mathbf{r}^h \mid \mathbf{y}^h)$ where $\mathbf{y}$ is the vector of hard-decisions, we obtain an alternative metric

$$\frac{f(\mathbf{r}^h \mid \mathbf{y}^h)}{f(\mathbf{r}^h \mid \mathbf{c}^h)}. \tag{3.24}$$

Using the usual factoring of the pdf's and taking logarithms, we obtain

$$\sum_{i=1}^{l} \log\left[\frac{f(r_i \mid y_i)}{f(r_i \mid c_i)}\right]. \tag{3.25}$$

It will be useful to define the *symbol likelihood-distance* as

$$D(y_i, c_i) = \left| \log \left[\frac{f(r_i \mid y_i)}{f(r_i \mid c_i)} \right] \right| . \tag{3.26}$$

(In (3.26) we have included the absolute value signs only so that $D(y_i, c_i) = D(c_i, y_i)$.) Note that $D(y_i, c_i) \geq 0$, and that equality is attained for $c_i = y_i$. Using (3.26) we see that (3.25) can be written as

$$D\left(\mathbf{y}^h, \mathbf{c}^h\right) = \sum_{i=1}^{l} D(y_i, c_i) \tag{3.27}$$

where $D\left(\mathbf{y}^h, \mathbf{c}^h\right)$ will be referred to as the *head likelihood-distance.* For heads of length n we obtain a *codeword likelihood-distance* $D(\mathbf{y}, \mathbf{c})$.

Where quantization is used the symbol likelihood-distance is as defined in (3.26), but with the pdf $f(r_i \mid \cdot)$ replaced by the probability $Pr(\ \mathcal{Q}(r_i) \mid \cdot\ )$, where $\mathcal{Q}(r_i)$ is the discrete random variable resulting from the quantization of r_i. For the case of hard-decision quantization on a binary–input symmetric–output memoryless channel to form a BSC it is easily shown that the codeword likelihood-distance $D(\mathbf{y}, \mathbf{c})$ reduces to a form that is equivalent to the Hamming distance $d_H(\mathbf{y}, \mathbf{c})$.

From the above discussion, we have that the decoder should prune those heads with the largest likelihood-distance $D\left(\mathbf{y}^h, \mathbf{c}^h\right)$, after defeating any heads that merge at a node in the trellis. From the above discussion on quantization, it is easy to see that this rule of first pruning the heads with the largest likelihood-distance is a generalization of the *path deletion axiom* [38] which states that whenever heads of the same length are being considered for pruning on a BSC, the one with the greatest Hamming distance to the received path is pruned first.

3.4 Contender Sifting

For partial trellis searches to be efficient it is implicit that there be an efficient method for *contender sifting*. Contender sifting refers to the selection of some the best contender metrics from among a larger number of contender metrics. The most common approach is to *sort* the contenders according to their metrics, and then retain those with the best metric. Alternatively, *selection* of the best (without regard to order within the best set) can be used. As pointed out in [29][39], the computational effort associated with contender sifting is significant, so that it severely limits the number of survivors that can be retained. In the stack algorithm [35] and its variants (e.g. [40]), it is common to reduce the computational effort of contender sifting by using approximate sorting; the contender metrics are sorted into quantized bins (buckets). The sorting is approximate since the bin contents are not sorted (i.e. the sorting is not a complete *bucket-sort* [41]). In breadth-first decoding such as the M algorithm, most approaches have used either sorting [42][39][43] or selection [39][44]. We will concentrate on breadth-first decoders, and in particular, the M algorithm, since it offers a decoding effort that is virtually constant, in contrast to metric-first searches [29].

We now illustrate the computational effort of various contender-sifting methods for the M algorithm. Much of our discussion follows that in [39]. Let $N_c(i,n)$ denote the number of comparisons used in sifting out the best i from among n contenders. For typical decoding applications such as rate 1/2 binary convolutional codes, or for binary linear block codes, it is appropriate to consider $N_c(M,2M)$. Figure 3.3 plots $N_c(M,2M)$ normalized by M, versus M, for various contender sifting methods. The specific methods shown in Figure 3.3 use insertion-sorting, merge-sorting, and Hadian and Sobel's selection algorithm [27].

The insertion-sort is particularly easy to implement but is suitable only for a very small number of inputs. Insertion sorting can be suitably modified to output only the best M of the $2M$ contenders.

This simple modification and the calculation of the worst-case number of comparisons are discussed in Appendix A. The worst-case number of comparisons normalized by M is, from Appendix A,

$$\frac{N_c(M,2M)}{M} \leq \frac{1}{2}(3M-1) \tag{3.28}$$

which is plotted in Figure 3.3.

Merge-sorting is considerably more efficient than insertion-sorting. The worst-case number of comparisons for merge-sorting is $O(n \lg n)$, where $\lg$ denotes $\log_2$ (versus $O(n^2)$ for insertion-sorting), and merge-sorting is an asymptotically optimum comparison-based sort [27][41][45]. Merge-sorting can be suitably modified to output only the best M of the $2M$ contenders. This simple modification and the calculation of the worst-case number of comparisons, for M being a power of 2, are discussed in Appendix A, from which we obtain

$$\frac{N_c(M,2M)}{M} \leq 2\lg M - 1 + 2/M. \tag{3.29}$$

Hadian and Sobel's selection algorithm uses

$$\min\{M + (M-1)\lceil \lg(M+2)\rceil,\ M - 1 + M\lceil \lg(M+1)\rceil\} \tag{3.30}$$

comparisons in the worst-case to find the M^{th} best of the $2M$ [27, pg 214]. Moreover, (3.30) gives the worst-case number of comparisons to partition the $2M$ contenders into the best and worst M, since any comparison-based selection (order-statistic) algorithm has carried out a sufficient number of comparisons to partition the input set into best and worst subsets [27][45]. For M a power of 2, the first term in (3.30) is the smaller[21], and this term normalized by M is

$$\frac{N_c(M,2M)}{M} = 1 + \left(1 - \frac{1}{M}\right)\lceil \lg(M+2)\rceil \qquad , M \text{ a power of 2} \tag{3.31}$$

[21] That the first term in (3.30) is smaller is verified as follows. For $M = 2^i$, $i \geq 0$, $\lceil \lg(2^i+2)\rceil = \lceil \lg(2^i+1)\rceil$. Let this quantity be denoted by A, we can write the first term in (3.30) as $M + (M-1)A$ and the second term as $M - 1 + MA$. The second term minus the first is $A - 1$ which is greater than or equal to zero for $i \geq 0$.

which is shown in Figure 3.3. Hence, Hadian and Sobel's selection algorithm takes $O(M \lg M) = O(\frac{n}{2} \lg \frac{n}{2})$ comparisons in the worst-case, or about one-half the number of comparisons of merge-sort.

Also shown in Figure 3.3 are the number of comparisons for the best known selection algorithms for small M [27, Table 1, pg 215]. For asymptotically large M, selection schemes exist which require a worst-case number of comparisons asymptotic to $10.86M$ [46] and $6M$ [47].

In the following sections we introduce a contender sifting method for breadth-first decoding that can retain a sorted set of M survivors with a worst-case number of comparisons that is linear in M, for any size of M. As will be described, the proposed method exploits an inherent ordering of the contender metrics. For decoding binary linear block codes, the proposed contender sifting method has a worst-case number of comparisons as shown as the lowest dotted line in Figure 3.3. For ease of description, the method will be presented for the M algorithm operating on a tree for a binary linear block code. Extensions of the contender sifting method for use in decoding convolutional codes, and other cases are then discussed.

3.4.1 Breadth-First Contender Sifting for Binary Linear Block Codes

Consider a code tree for a binary linear block code, with one channel-bit per branch[22]. For ease of illustration we assume that likelihood-distance is used as the decoding metric.

A typical stage in the operation of the M algorithm is illustrated in Figure 3.4. The decoder begins with a set S of M survivors and extends each survivor to the next depth to form a set C of $2M$ contenders, as shown in Figure 3.4(a) for the case of an information-bit. (The case of a

[22] Throughout this thesis, we make the natural assumption that there are no all zero columns in the code's generator matrix, so that there are no 'wasted' symbol positions that carry no information.

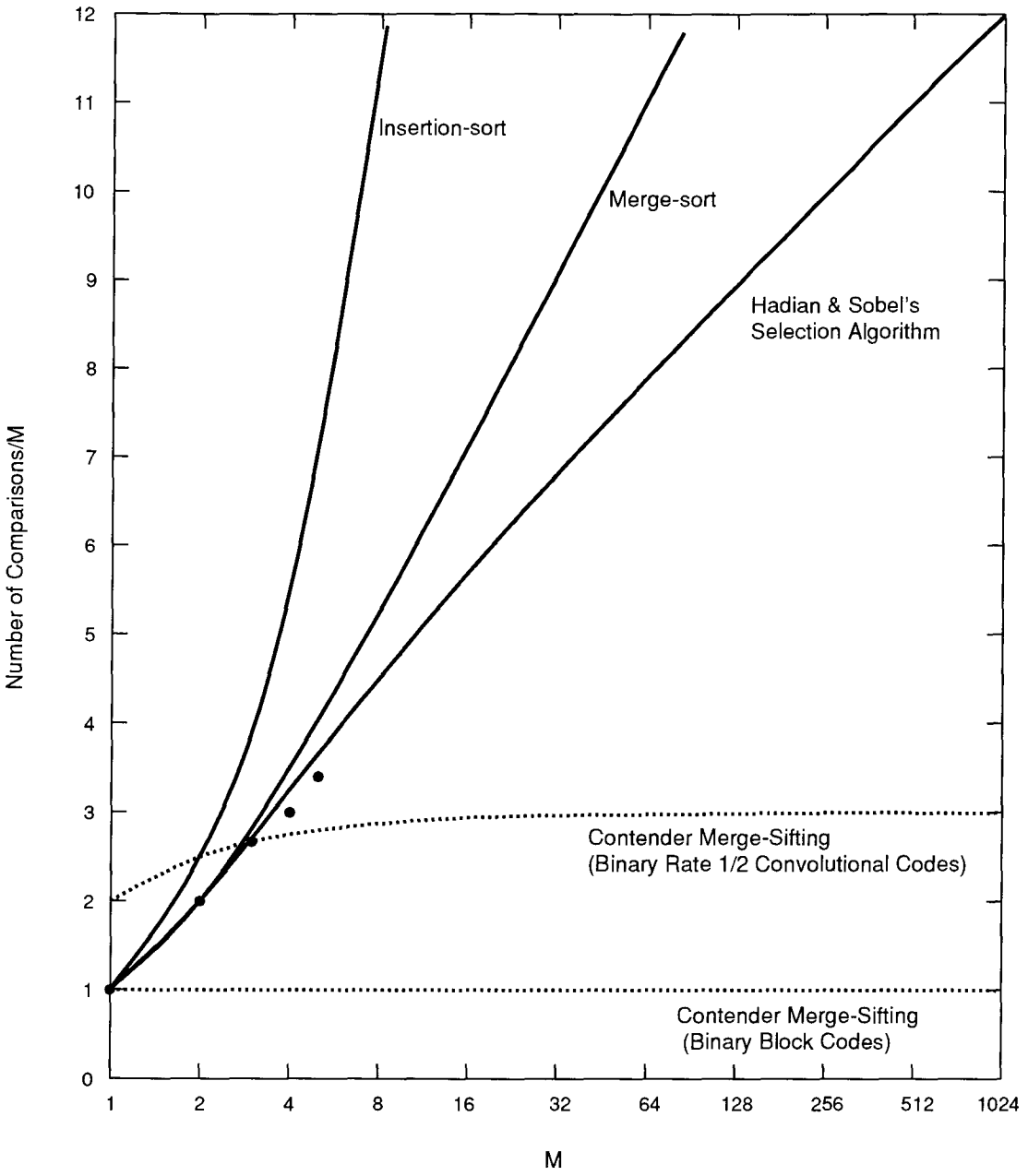


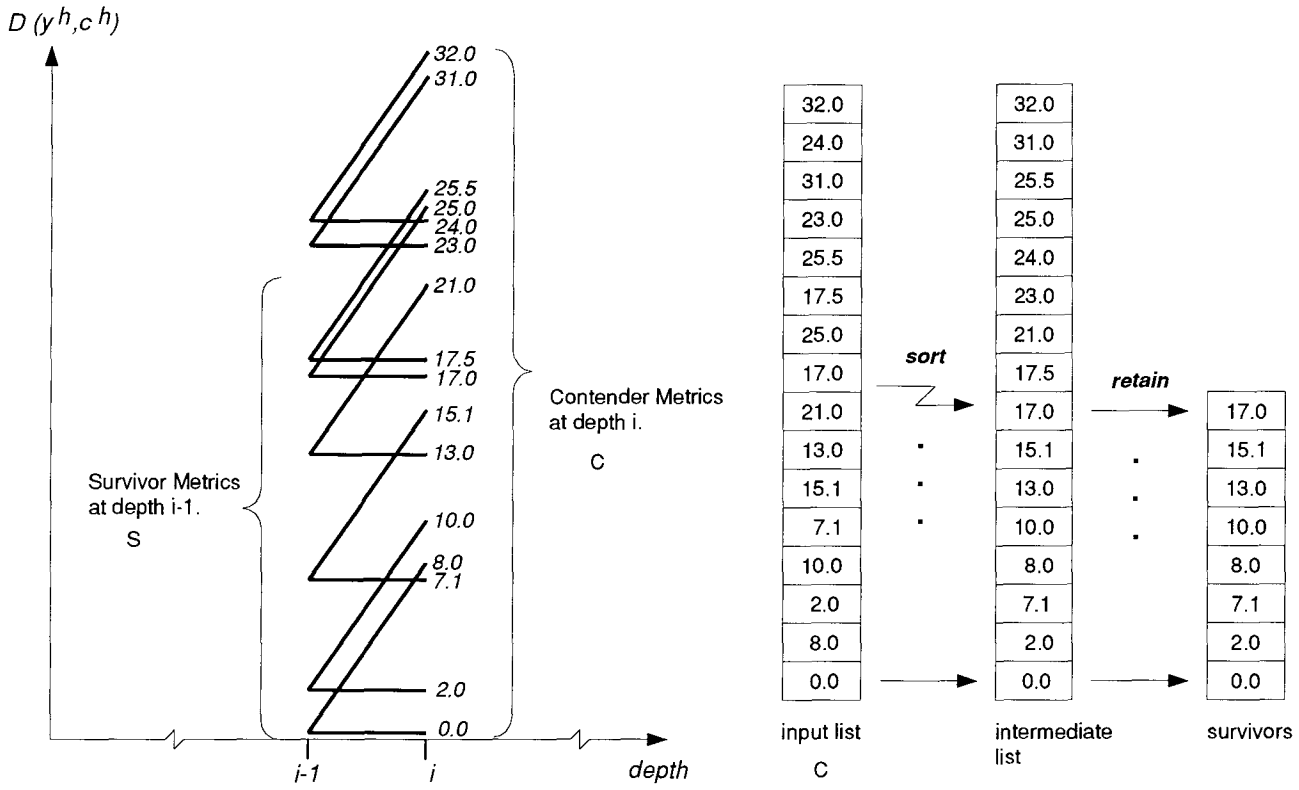
Figure 3.3 Normalized Number of Comparisons for Sifting M from $2M$ Contenders
The number of comparisons $N_c(M, 2M)$ (normalized by M) sufficient to sift out the best M of $2M$ contenders is plotted against M for various contender sifting methods. Simple modifications to insertion-sorting and merge-sorting were used to provide only the best M from $2M$ contenders. Comparison-based selection (order-statistic) methods, such as Hadian and Sobel's shown here, inherently partition the $2M$ contenders into the best M and the worst M. Also shown are the best known results for small M ($M = 2, 3, 4, 5$). The dotted lines correspond to the proposed contender *merge-sifting* method.

constraint-bit will be discussed later.) Standard contender sifting schemes operate on the list of $2M$ contender metrics shown in Figure 3.4(b), with the result being a list of M survivors (which may be either sorted or unsorted).

The efficiency of the proposed contender sifting method arises from a particular organization of the contenders within each decoding stage. Figure 3.5 illustrates a reorganization of the contender metrics into two *contender-subsets*. The first subset, labeled C_0 in Figure 3.5, consists of the M contender metrics that were formed by extending each survivor with the '0' bit. The second subset, labeled C_1, consists of the other M contenders that were formed by extending each survivor with the '1' bit. Of paramount importance is that each contender-subset is an sorted set, as shown in Figure 3.5. The ordering within the contender-subsets requires no sorting effort — the ordering is a consequence of our method for forming the contender subsets from an ordered set of survivors. For example in Figure 3.5, since bit '0' happens to be the hard-decision, each contender's metric in C_0 is identical to those in S (due to the fact that the likelihood-distance for the hard-decision bit is zero), and thus the ordering inherent in S is preserved in C_0. In C_1, each contender metric is formed by addition of an identical likelihood-distance to each survivor metric, so that the ranking inherent in S is also preserved in C_1. This organization of contenders enables the decoder to maintain a sorted list by *merging* the two sorted contender-subsets. Merging of two ordered sets of size M is very simple compared to complete sorting of a single set of size $2M$ [27], as the merging can be accomplished with $2M - 1$ comparisons. In fact, since we seek only the best M of the $2M$, the decoder can accomplish this in M comparisons by *partially* merging the two contender-subsets until the required number M have been found. This partial merging can be done in M comparisons, so we have

$$\frac{N_c(M, 2M)}{M} = 1 \tag{3.32}$$

which is the lowest dashed line in Figure 3.3.

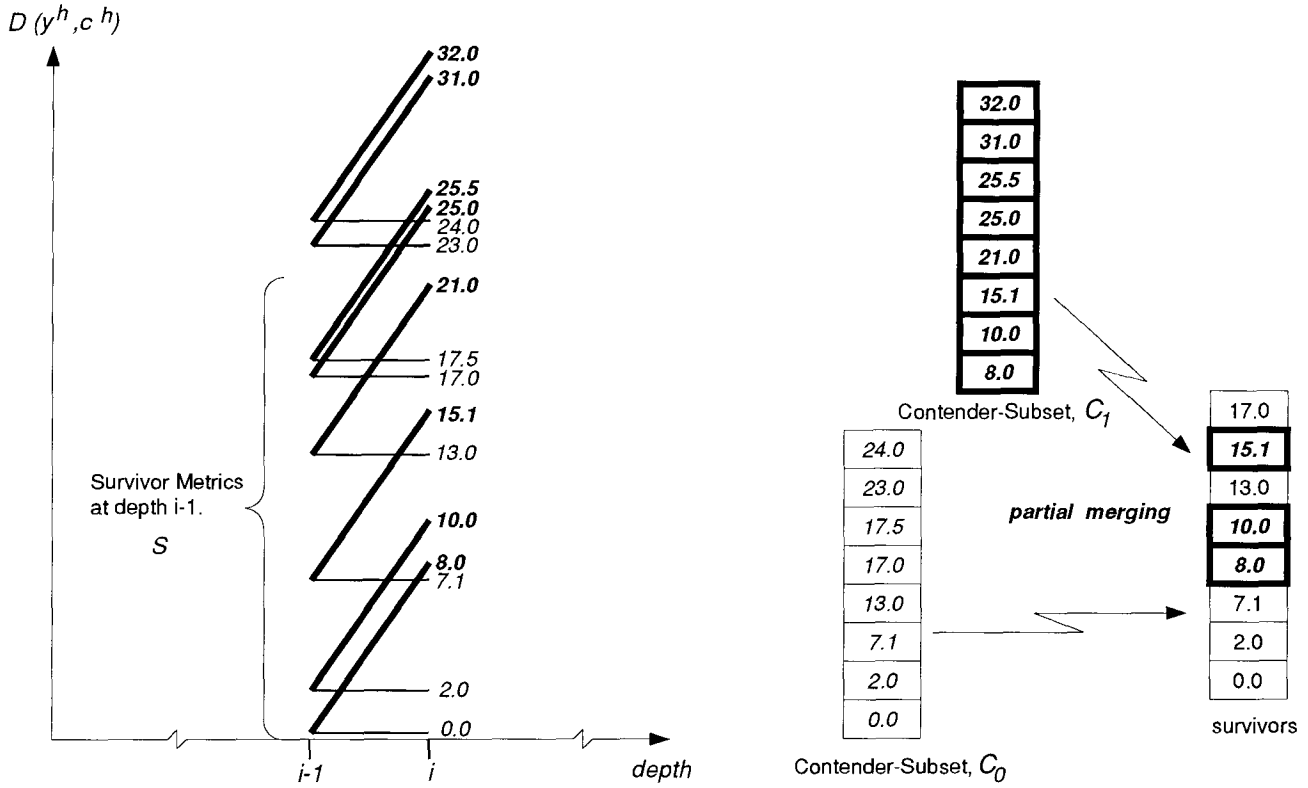


(a) Formation of the Contender Metrics.

(b) Contender Sifting using Sorting.

Figure 3.4 Contender Formation and Sifting

The formation of the contender metrics is shown in (a) for extension of the survivors at depth $i-1$ with an information-bit at position i. Only the branch likelihood-distances for position i are shown. The $M = 8$ survivor metrics at depth $i-1$ are updated to form $2M = 16$ contender metrics at depth i. In (b), the $2M$ contender metrics are shown in a list, which is then processed by the decoder to retain the best M. For simplicity we show complete sorting of the contenders, which is then followed by retention of the best half.



(a) Formation of the Contender Metrics.

(b) Contender Merge-Sifting, using Contender-Subsets and partial merging.

Figure 3.5 Contender-Subset Formation and Contender Merge-Sifting

In (a), during the extension of the survivors, the set of contender metrics is partitioned into two *contender-subsets.* according to whether the extended bit is a '0' or a '1'. In (a), the '1' extensions are indicated with darker lines than the '0' extensions (we have assumed, without loss of generality, that the '0' bit is the hard-decision.). In (b), these contender-subsets are shown as lists labeled C_0 and C_1. This choice of contender-subsets results in each subset being an ordered set, so that partial merging can be used instead of sorting. The result is an ordered list of M survivors.

The above discussion outlines the main principle of the proposed contender sifting method, which we shall refer to as *contender merge-sifting* (CMS). As will be discussed shortly, the basic contender merge-sifting method can be applied for constraint-symbols, as well as for other decoding applications.

Decoding Systematic Binary Linear Block Codes

The basic contender merge-sifting method discussed above arms us with a technique that is sufficient to implement M algorithm decoding of systematic binary linear block codes.

For codes in systematic form the M algorithm need only sift contenders where the number of contenders exceeds M. Let l_M denote the first depth where the number of contenders exceeds M. Contender sifting will then occur at depths l_M through k. The M algorithm has no need to sift contenders for the final $n-k$ depths, since the final $n-k$ depths are all constraint depths (where each extension of M survivors results in M contenders). To implement the M algorithm we can utilize contender merge-sifting as follows. For the depths 1 through $l_M - 1$, we use contender merge-sifting with *complete* merging of the contender-subsets, to maintain an sorted set of $2^1, 2^2, \ldots, 2^{l_M-1}$ survivors at the depths $1, 2, \ldots, l_M - 1$, respectively. This ensures that the set of survivors at depth $l_M - 1$ will be sorted, so that merge-sifting can be used at subsequent depths. At depths l_M through k, the decoder uses contender merge-sifting with *partial* merging of the contender-subsets, as shown in Figure 3.5, to maintain a sorted set of M survivors. At depths $k+1$ through n, no contender sifting is needed. The output codeword is chosen as the best among the M final survivors.

The total number of metric comparisons used for the CMS implementation of the M algorithm, for this systematic code case, can be upper bounded as follows. For the initial depths $1, 2, \ldots, l_M - 1$ where the number of contenders is less than M, the number of comparisons to maintain the ordered

set of $2^1, 2^2, \ldots, 2^{l_M - 1}$ survivors using complete merging is at most[23]

$$\sum_{i=1}^{l_M - 1} \left(2^i - 1\right) = 2^{l_M} - l_M - 1 \tag{3.33}$$

where the depth l_M is given by

$$l_M = \lfloor \log_2 (M) \rfloor + 1. \tag{3.34}$$

For the depths l_M through k, the maintenance of the ordered set of M survivors can be done using partial merging of the contender subsets, with M comparisons for each of these depths, so that

$$M(k - l_M + 1) \tag{3.35}$$

comparisons can accomplish the contender-sifting for these depths. The total number of comparisons is (3.33) plus (3.35), plus $M - 1$ comparisons to choose the best metric at depth n, giving

$$N_c \leq M(k + 2 - l_M) + 2^{l_M} - l_M - 2. \tag{3.36}$$

A simpler upper bound on N_c can be obtained by replacing the number of comparisons in the preceding analysis for the depths $2, 3, \ldots . L_1 - 1$, with M. Now, as in the preceding analysis, at the first depth we count one comparison, and to this we add the upper bound of M comparisons for each of the depths $2, 3, \ldots, k$, and lastly add in the $M - 1$ final comparisons at depth n, to give

$$\begin{aligned} N_c &< 1 + M(k - 1) + (M - 1) \\ &= Mk\,. \end{aligned} \tag{3.37}$$

[23] The reason that (3.33) is an upper bound is that we have assumed that each of the first $l_M - 1$ positions is an information position. This is a natural assumption since typically M is small, and for $M \leq 2^{d_{min} - 1}$ each of the first $l_M - 1$ positions are certainly information-positions.

Decoding Non-Systematic Binary Linear Block Codes

For decoding non-systematic codes the M algorithm again carries out contender sifting only at the information positions. To implement the M algorithm the decoder can utilize contender merge-sifting in a similar manner as above, except that it maintains the sorted survivor set until the last information-position. Let the last information-position be denoted as K. To maintain the sorted survivor sets at constraint-positions before K, contender merge-sifting works in a fashion similar to that shown in Figure 3.5, in that the contender-subsets are defined in the same way, but the difference being that there are no longer M contenders in each subset. The number of contenders in each subset may vary from 0 to M, with a total of M contenders shared between them. Since in this case we want to retain all M contenders, the decoder should completely merge the contender-subsets.

The number of comparisons used in contender merge-sifting at a constraint-position can be upper-bounded as follows. We use the same merging method as was used for the information-positions (where there were equal numbers of contenders in each contender-subset). The largest number of comparisons will occur for the case where there is only a single contender in one subset, say in C_1, and its metric is higher than any of the $M-1$ metrics in C_0. Thus, the largest number of comparisons will be $M-1$. This will account for an additional $(K-k)(M-1)$ comparisons over that of the systematic case, given by (3.36), so that we have

$$N_c \leq M(K+2-l_M) + 2^{l_M} - (K-k) - l_M - 2. \tag{3.38}$$

For any linear block code with minimum distance d_{min}, the maximum value that K can have is $n-(d_{min}-1)$[24], so that an upper bound on (3.38) is

$$N_c \leq M(n+3-d_{min}-l_M) + 2^{l_M} - (n-k-d_{min}) - l_M - 3$$

[24] This follows from the fact that any $d_{min}-1$ symbols of a linear block code can be chosen as parity symbols.

$$= M(k + \Delta + 2 - l_M) + 2^{l_M} - l_M - 2 - \Delta \tag{3.39}$$

where $\Delta \triangleq n - k + 1 - d_{min}$ is the amount by which d_{min} is less than the Singleton bound.

Alternatively, a simpler (but looser) upper bound can be formed by adding in the extra $(K - k)(M - 1)$ comparisons (arising from the non-systematicity) to the upper bound for the systematic case (3.37), to obtain

$$N_c < Mk + (K - k)(M - 1). \tag{3.40}$$

Finally, using the fact that $K \leq n - (d_{min} - 1)$ in (3.40) gives

$$\begin{aligned} N_c &< Mk + (M - 1)[n - k - (d_{min} - 1)] \\ &= Mk + (M - 1)\Delta \end{aligned} \tag{3.41}$$

as a simple upper bound on the number of comparisons to decode any binary linear block code using contender merge-sifting to implement the M algorithm.

Let us now return to Figure 3.3, and consider the worst case number of comparisons used in contender merge-sifting in decoding any binary linear block code. From our earlier discussion on decoding systematic codes, we had that there were at most M comparisons at any depth. From the discussion on decoding non-systematic codes, we had that there were at most $M - 1$ comparisons at any constraint position. Hence, the maximum number of comparisons at any depth is upper bounded by M, so that the lowest dotted line in Figure 3.3 is an upper-bound on the normalized number of comparisons at any depth, for M algorithm decoding of any binary linear block code.

3.4.2 Rate 1/n Binary Convolutional Codes

Here we show how contender merge-sifting can be applied to breadth-first tree decoding of rate 1/n binary convolutional codes. We will first consider rate 1/2 binary convolutional codes generated by an encoder of the form shown in Figure 3.6.

In Figure 3.6, each information-bit is shifted into the encoder shift register to produce two channel-bits. The shift register connections to the two sets of adders are specified by the generator polynomials $g_1(x) = g_{10} + g_{11}x + \cdots + g_{1m}x^m$ and $g_2(x) = g_{20} + g_{21}x + \cdots + g_{2m}x^m$. The convolutional code trellis will have $s = 2^m$ states, where

$$m = \max \{ \deg [g_1(x), g_2(x)] \} \tag{3.42}$$

is referred to as the constraint length of the code [48]. For example, constraint length 6 (64 state) codes are widely used in practice, with VLSI Viterbi algorithm decoders available.

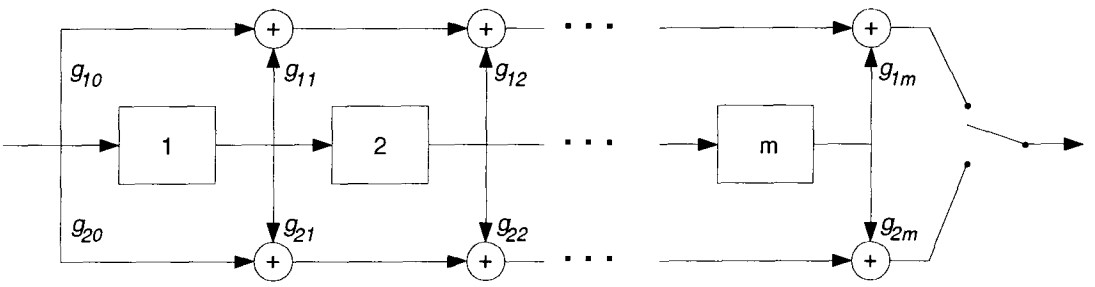


Figure 3.6 A Rate 1/2 Convolutional Encoder

A rate 1/2 binary convolutional code tree for the encoder in Figure 3.6 is almost invariably drawn with 2 channel-bits per branch. An alternate way to view the tree is to redraw each 2–bit branch as a pair of consecutive single-bit branches. In order to implement the M algorithm using contender merge-sifting, we first consider how the M algorithm's operation on the original tree can be viewed on the new tree. On the original tree the M algorithm prunes after each 2–bit branch[25]. This is equivalent, on the single-bit per branch tree, to not pruning after the first bit of the pair, and then pruning after the second bit. This can be accomplished using contender merge-sifting, as follows. Using the first bit of the pair, the decoder forms contender-subsets and merges them to retain all of the 2M contenders as survivors. At the second bit of the pair, contender-subsets are formed and partially merged to retain the best M contenders as survivors.

The number of comparisons used in this sifting procedure is the sum of the $2M-1$ comparisons for the complete merging, plus the M comparisons in the worst case for partial merging, giving

$$\begin{aligned} N_c &\leq 2M - 1 + M \\ &= 3M - 1 \ . \end{aligned} \tag{3.43}$$

The number of comparisons normalized by M is then $N_c/M \ \leq \ 3 - 1/M$ which is shown in Figure 3.3. It can be seen that CMS is very efficient compared to the other schemes, for all but the smallest M.

The above procedure generalizes for rate $1/n$ codes as follows. First, the n-bit per branch trellis is viewed as a concatenation of n single-bit branches. At the first bit, complete merging of the two contender-subsets of size M is used to retain $2M$ sorted contenders. At each of the next $n-2$ bits, complete merging of contender-subsets is again used, to retain $2M$ sorted contenders.

[25] For simplification, we assume that the decoder depth is not near either end of the tree or trellis, where no pruning is required. For the initial depths of the tree or trellis (where the number of contenders is less than M) the decoder can use complete merging as discussed for the binary block code case. For the final depths of the tree or trellis (where there are only constraint positions) no sifting is required.

At the final bit (of the n) partial merging of the contender-subsets is used to retain only the best M as survivors. The number of comparisons used in this n-step sifting procedure is then

$$\begin{aligned} N_c &\leq (2M-1) + (n-2)(2M-1) + M \\ &= (n-1)(2M-1) + M \\ &= M(2n-1) - (n-1) \end{aligned} \tag{3.44}$$

to maintain a sorted survivor list at each depth of the original n–bit per branch trellis.

3.4.3 Discussion

It should be clear that CMS is not restricted to *linear* tree codes. The essential requirement to use CMS is that the contender metrics be partitioned into sorted subsets, so that merging can be used. This in turn requires branches at the same depth of the tree to share a pool of branch metrics. Non-linear codes can have an unequal number of contenders in each contender-subset, which is similar to the case already considered for constraint-positions of linear codes.

We emphasize that although we have used the head likelihood-distance as the decoding metric, the use of CMS is not restricted to this metric. Any metric can be used provided that the contender metrics can be partitioned into sorted subsets. As stated above, this requires branches at the same depth of the tree to share a pool of branch metrics. For example, it is common to use the metrics $\log[f(r_i \mid 0)]$ and $\log[f(r_i \mid 1)]$ when decoding binary block codes, and each branch at depth i of the tree utilizes one of these two metrics.

With regard to trellis decoding, we comment that if so desired, the merging of contenders at trellis nodes can be used with contender merge-sifting. Competition among heads that merge at the trellis nodes will reduce the number of entries in the contender-subsets, and this does not destroy the ordered property of the each contender-subset.

Contender merge-sifting is readily applicable to breadth-first bidirectional decoding of block or convolutional codes. In bidirectional searches, a block (or convolutional code *frame*) of symbols is searched from both ends of a trellis, or from two code trees [49][50][51].

Finally, we comment that CMS is applicable to the important class of *punctured* convolutional codes. Perhaps the most common example of such a code is that of puncturing (i.e. not transmitting) certain bits in a rate 1/2 code, which generates a code of rate greater than 1/2 [18]. Punctured convolutional codes enable the use of one encoder and decoder (with very minor alterations) for a range of code rates. This is especially useful in automatic repeat-request (ARQ) systems employing incremental redundancy [20], where *rate-compatible* punctured convolutional codes are attractive [19]. In such applications CMS can be easily utilized, since at a punctured bit the relative metrics of the contenders will be unchanged. Consequently, the formation and merging of the contender subsets is simplified and will require no metric comparisons.

Chapter 4

Reconfigurable Trellis Decoding

To find an efficient technique which, given a received word, can generate a set of codewords that will with high probability contain that word which is most likely: This is the central problem.

G.D. Forney [5]

THE efficiency of partial trellis searches arises from the judicious selection of a small set of survivors. If a sufficient number of survivors are wisely chosen, the increase in probability of error over that of a ML decoder will be small. In this chapter we introduce a class of techniques referred to as Reconfigurable Trellis Decoding (RTD), or RT decoding, that can retain relatively few survivors at the cost of a negligible increase in error probability.

RT decoding is not simply a partial trellis search algorithm. In fact it can utilize well known partial tree or trellis search algorithms. The essence of RT decoding is to carry out the search using a different trellis for the code. The trellis used for decoding is dependent upon the reliability of

the received data, so that it is determined 'on-the-fly'. The trellis used is a 'reconfiguration' of the transmitted code trellis, in that it corresponds to a code with its symbol positions reordered according to their reliabilities.

In Section 4.1 we review non-trellis based and trellis based soft-decision decoders for linear block codes. In Section 4.2 we explain the concept of RT decoding in detail. The number of survivors required to attain 'near-ML' performance will depend on the channel and the code. On soft-decision channels such as the AWGN channel with binary PSK, there can be a significant reduction in the number of survivors required. An even larger reduction is attained for an erasure channel, where $d_{min} - 1$ or fewer erasures can always be corrected for *any* linear block code with only a single survivor.

In Section 4.3 we extend the *Uniform Error Property* [52] of ML decoding of linear codes on a binary-input symmetric-output memoryless channel to the more general case of pruning decoding. This result is used in later sections.

In Section 4.4 we introduce the *pruning impact*, which is the increase of the probability of error of a pruning decoder with respect to a ML decoder. An upper bound on the pruning impact is found for use in later sections.

In Section 4.5 the pruning impact of RT decoding using the M algorithm is estimated. An *order statistic channel* (OSC) model is introduced and used to facilitate the assessment of the pruning impact. The pruning impact of the RT-M algorithm is found using a simple search procedure that is independent of the detailed structure of the code. Examples are given to compare the estimated word error rate of the RT-M algorithm to simulation results.

In Section 4.6 the computational effort of RT-M decoding is discussed. Tables of formulas are

presented that give the number of metric operations (comparisons, additions) and the number of binary-vector operations used in carrying out the search. Plots of the number of operations against coding gain are presented for some example codes on an AWGN channel with binary PSK and on a Rayleigh faded AWGN channel with binary NCFSK.

In Section 4.7 we review some of the main results of the previous sections and discuss RT decoding in terms of its resiliency to bursts of errors. Some other aspects of RT decoding are briefly reviewed, including its decoding delay and its computational overhead.

4.1 Soft-Decision Decoders for General Linear Block Codes

The majority of soft-decision decoding schemes for block codes do not utilize trees or trellises. Here we discuss some of the main schemes that have been proposed for ML decoding, or 'near-ML' decoding, of linear block codes. One of the earliest ML soft-decision decoding algorithms is Wagner decoding [53], which is applicable only to the simple $(n, n-1)$ single parity check codes. The Wagner decoding algorithm consists simply of inverting the least reliable bit if the initial check on parity was not met. Note that hard-decision decoding of this code cannot correct any errors, while soft-decision Wagner decoding on the AWGN channel with binary PSK will approach $d_{min} - 1 = 1$ bit error correction with increasing SNR [6].

In 1966, Forney introduced *Generalized Minimum Distance* (GMD) decoding [5][10], which provided a way to utilize an errors and erasures decoder for soft-decision decoding. In GMD the received word is modified by erasing a number of the least reliable symbols, and it is then fed to an errors and erasures decoder to generate a candidate codeword. Several candidate codewords are generated (at most about $\lfloor d_{min}/2 \rfloor$) by erasing a successively larger number of the least reliable symbols. This procedure is also referred to as successive erasures decoding (SED) [54]. The

'generalized distance' is used in a stopping rule to halt the generation of candidate codewords, and can thus lower the average decoding effort. Various modifications have been suggested to the Generalized Distance metric and stopping rule, including [54][55][56].

In 1972, Chase [6] introduced a method that utilizes a hard-decision decoder to generate candidate codewords. A candidate codeword is generated by first perturbing some bits of the received word and then feeding it to a hard-decision decoder. By using a number of such perturbations and subsequent decodings, the resultant set of candidate codewords are then compared using a soft-decision metric. The main perturbation schemes discussed in [6] alter the least reliable bits. Since the perturbation of the received word amounts to 'guessing' the errors, the technique is sometimes referred to as 'Chasing' [57].

There are many other examples of soft-decision decoding schemes that similarly require either a hard-decision decoding algorithm [58][59][60], or that detailed structure of the code be exploited e.g. [61][62][63][64][65].

A class of decoding schemes that can in principle be applied to any linear block code, and without the need for an errors/erasures decoder or a hard-decision decoder, is information set decoding. The central concept of information set decoding is as follows. If we can find a set of k independently specified symbols (an information set) that is error free, then the remaining $n-k$ symbols (a parity set) will be correctly determined by the constraints of the code. Since any error pattern that occurs in these $n-k$ parity positions will not alter the decoded output, these errors are said to be 'trapped'. A number of information sets are utilized to attempt to trap a specified number of errors. These information sets can be precomputed, or selected during decoding by some deterministic or random method. Specialized versions of information set decoding (e.g. [66][67]) generate a restricted number of information sets by carrying out automorphic (code-preserving)

permutations of the symbols. To consider the decoding effort for general linear (n, k) block codes, let $N(n, k, t)$ denote the minimum number of information sets required to trap t errors. It is known [68][69] that

$$N(n,k,t) \geq \left\lceil \frac{n}{n-k} \left\lceil \frac{n-1}{n-k-1} \cdots \left\lceil \frac{n-t+1}{n-k+t+1} \right\rceil \cdots \right\rceil \right\rceil. \tag{4.1}$$

A more readily computed, albeit looser, lower bound is [69]

$$N(n,k,t) \geq \frac{\binom{n}{t}}{\binom{n-k}{t}}. \tag{4.2}$$

In (4.2), the numerator corresponds to the number of t–tuples to be trapped, and the denominator corresponds to the number of t–tuples which can be trapped by each information set. Now suppose that for soft-decision decoding that we wish only to trap up to $d_{min} - 1$ errors, so that the decoder can attain the asymptotic ML error probability [6] [70]. Setting $t = d_{min} - 1$ in (4.2) yields

$$N(n,k,d_{min}-1) \geq \frac{\binom{n}{d_{min}-1}}{\binom{n-k}{d_{min}-1}}. \tag{4.3}$$

From this we see that for codes with $d_{min} - 1$ approaching the Singleton bound of $n - k$ that the denominator of (4.3) approaches unity, so that the ability of any information set to trap multiple t–tuples is entirely lost. This bodes poorly for soft-decision decoding of large minimum distance codes using pure information set decoding.

An alternative to pure information set decoding is to use one or more information sets and search through some error patterns, e.g. [71]. To further reduce the number of information sets considered, one can guide the choice of information sets by utilizing the soft-decision bit reliabilities [69][72]. As will be discussed later, RT decoding utilizes this approach and combines it with trellis decoding.

Trellis based soft-decision decoding of general linear block codes has received relatively little attention in the literature. The trellis constructions of Wolf [8], Massey [9], or Forney [16], can

be used with the Viterbi algorithm to attain ML decoding. For some codes with specialized structure, particularly efficient trellis based decoders can be developed [16]. For general linear block codes, a possible benefit of using trellis or tree representations is that decoding algorithms originally conceived for decoding convolutional codes may be applicable to decoding block codes. The various breath-first, depth-first, and metric-first search strategies for convolutional codes, e.g. [29][30][33][35][40][73][74], may be useful in decoding block codes. However, few results have been published on the application of such methods to decoding block codes. Matis and Modestino [22] proposed the use of a modified M algorithm to decode a linear block code trellis. In this scheme the number of trellis branches searched is reduced by using a limit of M survivors, and, for a specified number of the most reliable information positions, by only extending branches that agree with the hard-decisions. This idea was later applied to a Rayleigh faded AWGN channel in [75]. In using convolutional codes the length of the code is sometimes truncated in order to strictly limit the number of errors incurred when the correct path is lost [52]. This is naturally achieved in using block codes. The finite length of block codes and truncated convolutional codes (which are effectively block codes) can also be exploited in bidirectional searches of the trellis [49] or tree [50][51].

4.2 The RT Decoding Concept

As discussed in Chapter 3, in a partial search of a trellis the decoder utilizes the currently gathered likelihood information to guide its subsequent exploration. The concept of Reconfigurable Trellis Decoding is to combine partial trellis searching with the fact that we may exploit alternative trellis representations of the code based on symbol position permutations. The decoder operates on an equivalent-code trellis that has its symbol positions in an order advantageous to decoding. After

decoding of the equivalent-code the symbols are simply restored to their original order. The RT decoder aims to exploit reliable symbols by repositioning them to improve the search efficiency.

Efficiency in partial trellis searches is attained by exploring heads that are most likely to be part of the correct (transmitted) path, or equivalently, by discarding those heads that are unlikely to belong to the correct path. At each decoding stage the decoder compares head metrics to discard those heads that are presently ranked as unlikely. Increased search efficiency would be attained if the *rank order* of head metrics rapidly converged with depth to their final values. In other words, efficient pruning would be facilitated if the influence of the metrics of the unexplored tails was insignificant. Since reliable symbol-positions have one symbol-hypothesis that is much more likely than its alternatives, and since unreliable symbol-positions have little distinction between alternative symbol-hypotheses, reconfiguring the symbol positions in a most reliable symbol first (MRSF) manner should increase the rate of convergence with depth of the rank order of head metrics. In other words, using MRSF ordering should enable the head exploration to rapidly gather and utilize the most significant branch-likelihood information.

The above heuristic argument is perhaps reasonable for tree decoding, but it is less plausible in the case of a trellis. With a trellis, symbol position reordering may vary the trellis dimensions, as discussed in Chapter 2. If the code had a compact trellis then symbol position reordering may significantly increase the trellis dimensions and the search effort saved by using such a compact representation may be lost. We remark that as discussed in Chapter 2, for a linear block code to have a compact trellis requires a smallest minimum distance $d = \min\left(d_{min}, d_{min}^{\perp}\right)$ significantly less than the corresponding Singleton bound. For codes with a large d relative to the corresponding Singleton bound the impact on the trellis dimensionality due to symbol position reordering will be small. For example, MDS codes will have no alteration of trellis dimensions due to symbol position

reordering, so that any reordering can be used without a dimensional penalty.

With partial searches the question arises as to whether the search should use a trellis or a tree. With partial searches of large dimensional trellises we are constrained in practice to retain only some manageable number of survivors, which will be small in relation to the total number of states. It has been found (for example, see [32]) that it is unlikely that such survivors merge, so that trellis decoding offers little reduction in the number of survivors. As well, with partial trellis searches of large trellises it is not feasible to store the entire trellis, so that it is necessary to *compute* the contender states. This results in the need to find matching states amongst a sizeable number of survivors, which can involve significant effort [32]. For these reasons, we are inclined to use tree decoding for large dimensional codes.

Despite the above practical reasons favouring tree decoding over trellis decoding, we prefer to use the designation Reconfigurable *Trellis* Decoding since it is clear that the technique can also be utilized on the less structured tree-representation of the code. We often use the abbreviation *RT decoding*, which can conveniently have either interpretation, with the particular meaning being clear from the context. In cases where possible confusion may arise, the appropriate full form will be used.

As mentioned above it may not be feasible to store the entire trellis or tree so that the contender states should be computed — only the explored portion of the trellis or tree is generated. To do this, one can use the simplified trellis construction methods discussed in Chapter 2, except that states are found only for those heads that are extended. For use with RT decoding we note that the reordering of the symbol positions implies a corresponding reordering of the columns of the parity check matrix $\mathbf{H}$ and the generator matrix $\mathbf{G}$. In this chapter, we will use the 'Method 2' simplified trellis construction (pg. 32) and reduce $\mathbf{H}$ to row echelon form prior to beginning the search.

An Example. We now present examples of decoding using RTD and decoding using the original trellis for a Hamming (7,4) code. With this small code our intent is only to illustrate the basic mechanics of RTD.

Assume that the transmitter sends the codeword

$$\mathbf{c}_{transmitted} = (1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1) \tag{4.4}$$

and that the original trellis is as shown in Figure 4.1. At the receiver, assume that the resulting symbol likelihood-distances are

$$(0.1 \quad 0.3 \quad 0.4 \quad 1.6 \quad 0.8 \quad 1.2 \quad 0.2) \tag{4.5}$$

(where, since the code is binary, we need only show the single nonzero likelihood-distance for each symbol position). As well, we assume that the vector of hard-decisions is

$$\mathbf{y} = (0 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 0). \tag{4.6}$$

A hard decision decoder would decode to the codeword nearest in Hamming distance to $\mathbf{y}$, which is $\mathbf{c}_{out,HD} = (0 \quad 0 \quad 1 \quad 1 \quad 1 \quad 1 \quad 0)$.

For simplicity we assume that decoding is performed using the M algorithm with M=1, so that at each depth only a single survivor is retained. Figure 4.1 shows the original code trellis with the explored heads indicated by dark branches, and with the explored head's likelihood-distances indicated at each depth. In this example the decoded codeword is $\mathbf{c}_{out} = (0 \quad 1 \quad 1 \quad 0 \quad 1 \quad 0 \quad 0)$ which, like the hard-decision decoder, is in error. For RT decoding we first reorder the bit positions according to their reliability. The most-reliable-symbol-first order of likelihood-distances is

$$(1.6 \quad 1.2 \quad 0.8 \quad 0.4 \quad 0.3 \quad 0.2 \quad 0.1). \tag{4.7}$$

The original parity check matrix is shown below along with the new parity check matrix formed by correspondingly reordering the columns. Also shown is the reduced reordered matrix, where the arrows at the top of the reduced matrix indicate the constraint positions.

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 1 \end{bmatrix} \quad \begin{bmatrix} 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 1 & 0 \end{bmatrix} \quad \begin{bmatrix} 1 & 1 & 1 & \downarrow 1 & 0 & \downarrow 0 & \downarrow 0 \\ 1 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 1 \end{bmatrix}$$

Figure 4.2 shows the resulting code trellis with the explored heads indicated by dark branches. The RTD output codeword is

$$\mathbf{c}_{out,RTD} = (1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1) \tag{4.8}$$

which agrees with $\mathbf{c}_{transmitted}$.

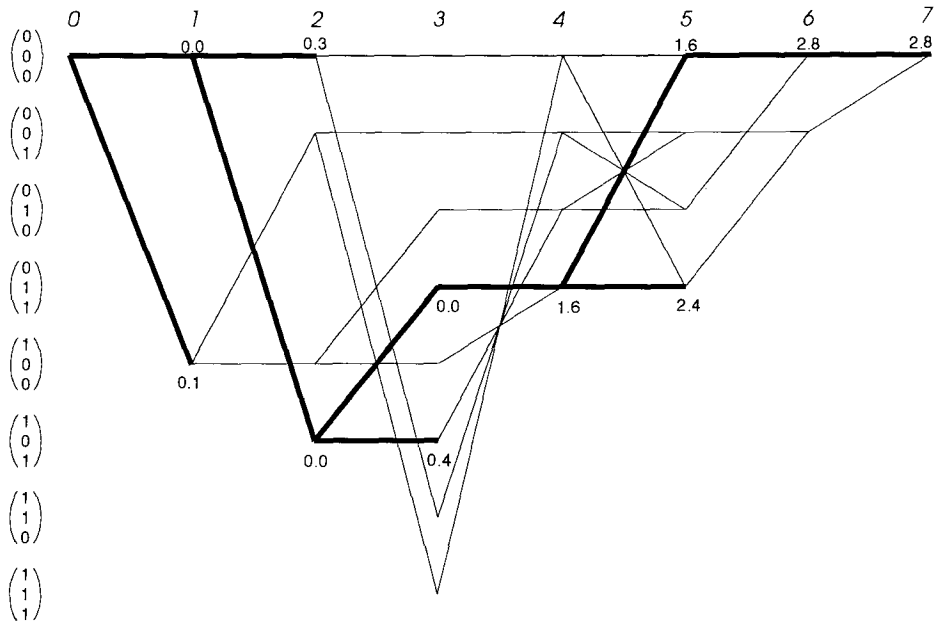


Figure 4.1 An Example of the M Algorithm Decoding a (7,4) Code
The dark branches indicate the explored trellis using the M algorithm with M=1. The output codeword as determined by the lowest likelihood-distance of the explored heads is c_{out} =(0 1 1 0 1 0 0).

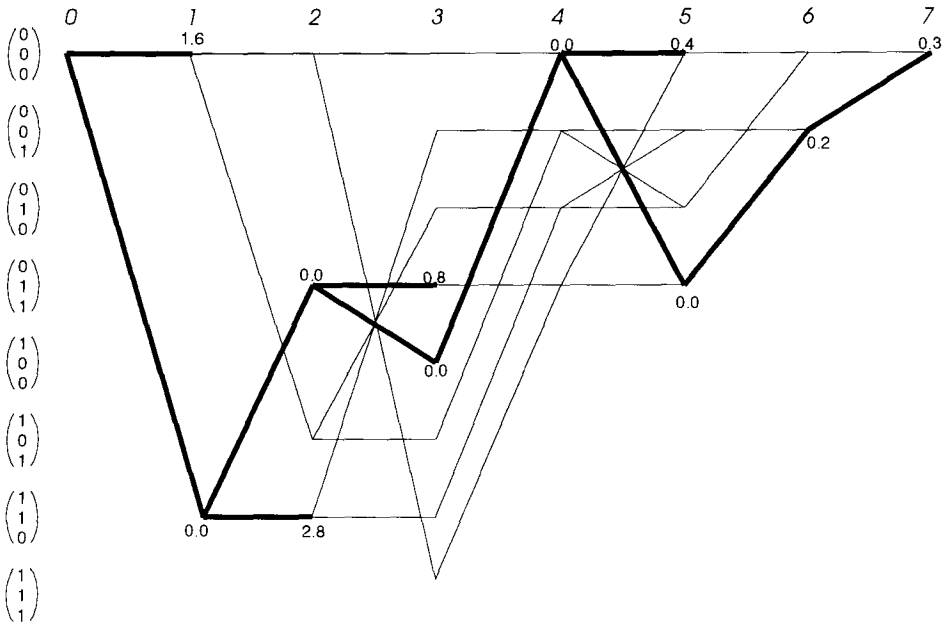


Figure 4.2 An Example of the RT-M Algorithm Decoding a (7,4) Code
The output codeword using the RT-M algorithm with M=1 is $c_{out,RTD}$ =(1 1 1 1 1 1 1).

This simple example also illustrates a key property of RT decoding. Consider the reordered vector of likelihood-distances (4.7) and note that since a smaller likelihood-distance implies a larger probability of error, we see that RT decoding will tend to collect channel errors into bursts in the tail of the trellis, regardless of the original error distribution. This can result in 'trapping' many errors in the tail (i.e. many errors will fall in the constraint positions in the tail), so that few survivors are required.

The number of errors that will be trapped by RT decoding depends on the nature of the channel. For example, if we are fortunate enough to have a channel that is extremely well approximated by an erasure channel, then RTD will collect all erasures in the tail. In the case of an MDS code on an erasure channel, if there are $n-k$ or fewer erasures they will all be in the final $n-k$ positions of the reconfigured trellis, and these final $n-k = d_{min}-1$ positions will be constraint positions — thus we can ensure that *ML decoding can be attained by retaining only one survivor*, regardless of the size of the code. Similarly, for non-MDS codes, we can be assured to correct any pattern of $d_{min}-1$ errors with only one survivor.

In fairness, the erasure channel represents the best possible case for RT decoding. It is easy to see that in the case of the BSC, there is no advantage to trellis reconfiguration. However, we are interested in more typical soft-decision channels, where we can gain some improvement in error probability compared to hard-decision decoding while at the same time being able to save trellis searching effort via RT decoding.

An Example of Decoding on an AWGN Channel

Here we present some simulation results for the binary (24,12) extended Golay code decoded using the M-algorithm, with and without reconfiguration, and the Viterbi algorithm. Figure 4.3 shows the codeword error rate (WER) versus signal-to-noise ratio (SNR[26]) in dB for binary phase-shift-keying (BPSK) on an additive white Gaussian noise (AWGN) channel. It is easy to show that the likelihood-distance for each bit is proportional to $|r_i|$, where r_i is the output of the demodulator for the i^{th} bit.

The simulation results shown in Figure 4.3 demonstrate a considerable reduction in the number of survivors when using RT decoding. For example, compared to hard-decision decoding, the M algorithm with 16 survivors performs slightly better, while the RT-M algorithm with only one survivor outperforms both. The RT-M algorithm with 8 survivors attains virtually the same WER as the M algorithm with 128 survivors. To be within 0.25 dB of the Viterbi algorithm at a WER of 10^{-2}, the RT-M algorithm needs about 8 survivors, while the M algorithm needs about 128. This compares with the smallest s (the maximum trellis dimension) attainable for the (24,12) code [17] of 9, which corresponds to 512 states.[27]

---

[26] Here the SNR is E_b/N_0, i.e. the ratio of the energy per information-bit to the one-sided power-spectral-density (PSD) of the AWGN.

[27] As discussed in [17], trellises with smaller dimensions can be drawn if one is willing to group multiple bits per trellis branch. In [16] a trellis for the Golay (24,12) code is constructed which facilitates an efficient decoding algorithm for this particular code. Later we will present a more detailed comparison of these approaches.

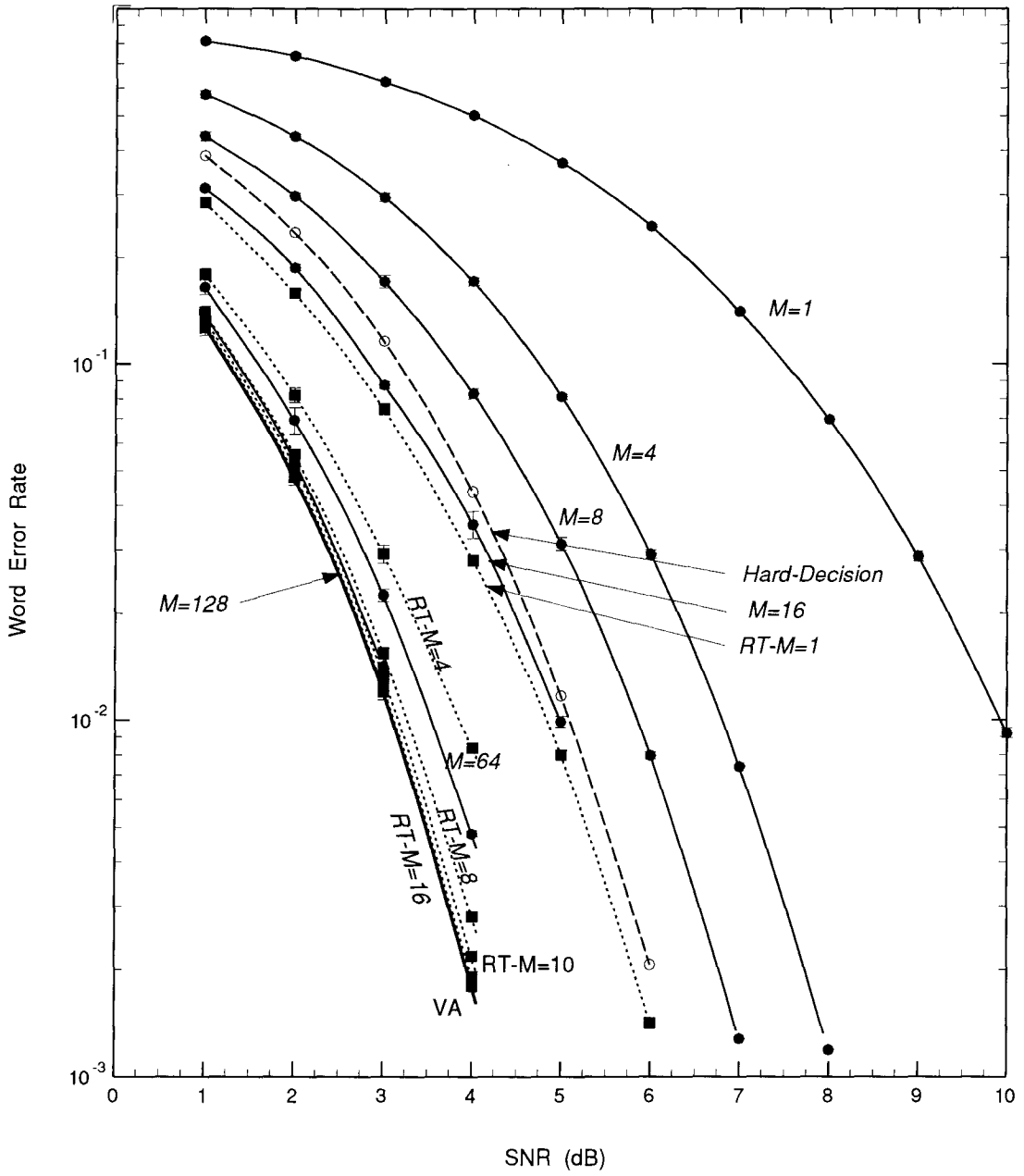


Figure 4.3 Word Error Rate using the M Algorithm, RT-M Algorithm and the Viterbi Algorithm
The word error rate (WER) is plotted vs. the SNR for the (24,12) Golay code on an AWGN channel with BPSK. The solid and dotted curves show the M algorithm and the RT-M algorithm word error rates, respectively. The error bars show 95% confidence intervals.

4.3 The Uniform Error Property for Pruning Decoders

In this section we discuss how the *uniform error property* of ML decoding of linear codes on an *binary-input symmetric-output* memoryless channel [52] can be extended to the more general case of pruning decoding.

A *binary-input symmetric-output* memoryless channel [52] has inputs labeled 0 and 1 and has an output that is symmetric in the sense that

$$f(r_i \mid 0) = f(-r_i \mid 1) \tag{4.9}$$

for all values of r_i. For example, the AWGN channel, the BSC, and any other symmetrically quantized AWGN channel can be shown to satisfy (4.9)[28] [52]. This symmetry can be used to show that the *uniform error property* [52] holds for linear codes with ML decoding, i.e. that

$$P_{e,ML} = P_{e,ML|\mathbf{c}_j} \qquad \text{, any } \mathbf{c}_j \in \mathbf{C} \tag{4.10}$$

where $P_{e,ML}$ is the error probability using ML decoding, and $P_{e,ML|\mathbf{c}_j}$ denotes the error probability given that a specific codeword $\mathbf{c}_j$ is transmitted. The uniform error property allows one to evaluate the probability of error for an ML decoder by assuming that any single codeword is sent, which for convenience can be taken to be the all zero codeword $\mathbf{c}_0$. One would expect that the uniform error property should hold for pruning decoders. This generalization of (4.10) is stated in the following theorem, which is proved in Appendix B.

Theorem 4.1 The error probability P_e of a pruning decoder, when decoding linear codes on a tree or a minimal trellis and assuming a binary-input symmetric-output memoryless channel, satisfies

$$P_e = P_{e|\mathbf{c}_j} \qquad \text{, any } \mathbf{c}_j \in \mathbf{C} \tag{4.11}$$

[28] A simple transformation of the output may be required so that (4.9) is satisfied. For example, suppose an AWGN channel is used with the signal constellation points for 0 and 1 at $r = 0$ and $r = A$, respectively. To satisfy (4.9), one needs only to shift r so that the signal points occur at $\pm A/2$.

where $P_{e|\mathbf{c}_j}$ denotes the error probability given that codeword $\mathbf{c}_j$ is transmitted. The pruning decoder is assumed to adhere to the decoding rule of minimizing (3.19) at each decoding stage, and in doing so it is assumed to treat each codeword as equiprobable.

4.4 The Pruning Impact

When decoding using pruning it is desirable to attain "near-ML performance". To quantify this we define the *pruning impact*, P_I, to be the increase in probability of error of a pruning decoder with respect to a ML decoder. Introducing the pruning impact will also facilitate the analysis, compared to using the error probability directly.

In any decoder, an error occurs when the correct path (CP) is not released as output; which in the case of a pruning decoder is due to the CP being pruned, or due to the most likely path of those remaining being chosen instead of the CP. The error probability can be written as

$$\begin{aligned} P_e &= \int Pr(\text{ CP not chosen } | \mathbf{r})\, f(\mathbf{r})\, d\mathbf{r} \\ &= \int Pr(\text{ CP pruned } \cup \text{ ML path} \neq \text{CP } | \mathbf{r})\, f(\mathbf{r})\, d\mathbf{r}. \end{aligned} \tag{4.12}$$

Forming a union bound on the error causing events in (4.12), we obtain

$$P_e \leq \int [Pr(\text{ CP pruned } | \mathbf{r}) + Pr(\text{ ML path} \neq \text{CP } | \mathbf{r})]\, f(\mathbf{r})\, d\mathbf{r}. \tag{4.13}$$

Let $P_{e,ML}$ denote the error probability of an ML decoder. We see that (4.13) is

$$P_e \leq \int Pr(\text{ CP pruned } | \mathbf{r})\, f(\mathbf{r})\, d\mathbf{r} \quad + \quad P_{e,ML} \tag{4.14}$$

so that

$$P_e - P_{e,ML} \leq \int Pr(\text{ CP pruned } | \mathbf{r})\, f(\mathbf{r})\, d\mathbf{r}. \tag{4.15}$$

The left hand side of (4.15) is precisely our definition of the pruning impact. Thus we have upper bounded the pruning impact P_I as

$$P_I \leq \int Pr(\text{ CP pruned} \mid \mathbf{r})\, f(\mathbf{r})\, d\mathbf{r}. \tag{4.16}$$

It is easy to verify that the upper bound (4.16) is reasonable. From (4.12) and (4.13) we note that the amount by which (4.16) will exceed P_I is precisely

$$Pr(\text{ CP pruned} \cap \text{ CP} \neq \text{ML path}). \tag{4.17}$$

However, this can be simply bounded via

$$Pr(\text{ CP pruned} \cap \text{ CP} \neq \text{ML path}) \leq P_{e,ML} \tag{4.18}$$

and we see that the looseness of (4.16) is upper bounded by $P_{e,ML}$.

It will be convenient to recast the upper bound (4.16) into a different form. Let $\widehat{P}_I$ denote the right hand side of (4.16), where the 'hat' indicates that this quantity is an upper bound.

We now assume that the channel is a binary-input symmetric-output memoryless channel [52]. Consequently, Theorem 4.1 applies, and it also implies that the pruning impact satisfies the uniform error property,

$$P_I = P_{I|\mathbf{c}_j} \qquad \text{, any } \mathbf{c}_j \in \mathbf{C} \tag{4.19}$$

where $P_{I|\mathbf{c}_j}$ denotes the pruning impact given that only a codeword $\mathbf{c}_j$ is transmitted. As well, the bound on the pruning impact (4.16) will satisfy a similar condition. Using this result in (4.16), gives

$$\widehat{P}_I = \int_{\mathbf{R}} Pr(\mathbf{c}_0 \text{ pruned} \mid \mathbf{r}, \mathbf{c}_0 \text{ transmitted})\, f(\mathbf{r} \mid \mathbf{c_0} \text{ transmitted})\, d\mathbf{r}. \tag{4.20}$$

Now

$$Pr(\mathbf{c}_0 \text{ pruned} \mid \mathbf{r}, \mathbf{c}_0 \text{ transmitted}) = \begin{cases} 0 & , \mathbf{c}_0 \text{ not pruned} \\ 1 & , \mathbf{c}_0 \text{ pruned} \end{cases} \tag{4.21}$$

so that we can rewrite (4.20) as

$$\widehat{P}_I = \int_{\mathbf{R}_0} f(\mathbf{r} \mid \mathbf{c}_0 \text{ transmitted})\, d\mathbf{r} \tag{4.22}$$

where $\mathbf{R}_0$ is the region of $\mathbf{R}$ where $\mathbf{c}_0$ is pruned. Equation (4.22) will be utilized in subsequent sections.

4.5 Approximate Analysis of Pruning Impact for the RT-M Algorithm

4.5.1 The Order Statistic Channel Model

As discussed earlier, a binary-input symmetric-output memoryless channel has outputs r_i and $-r_i$ such that $f(r_i \mid 0) = f(-r_i \mid 1)$. Consequently, we also have $f(-r_i \mid 0) = f(r_i \mid 1)$. These symmetric likelihoods can be conveniently represented by a transition diagram as shown in Figure 4.4, where without loss of generality we have let $r_i = \rho_i \geq 0$ correspond to the larger likelihood of $f(r_i \mid 0)$ and $f(-r_i \mid 0)$.

It will be convenient to work with conditional transition probabilities. Conditioning on the output being either ρ_i or $-\rho_i$, and given that 0 was sent, the crossover probability is

$$\begin{aligned} Pr(-\rho_i \mid \ 0, |\rho_i|) &= \frac{f(-\rho_i, |\rho_i| \ \mid 0)}{f(|\rho_i| \ \mid 0)} \\ &= \frac{f(-\rho_i \mid 0)}{f(\rho_i \mid 0) + f(-\rho_i \mid 0)} \\ &= \frac{f(\rho_i \mid 1)}{f(\rho_i \mid 0) + f(\rho_i \mid 1)} \end{aligned} \tag{4.23}$$

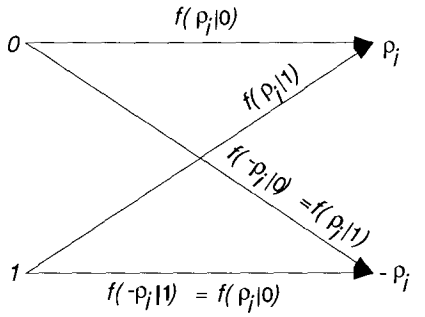


Figure 4.4 A Symmetric Pairing of Outputs from a Binary-Input Symmetric-Output Channel

where we have used the fact that $f(-\rho_i \mid 0) = f(\rho_i \mid 1)$. Similarly, the crossover probability given that 1 was sent and that ρ_i or $-\rho_i$ occurred, is

$$\begin{aligned} Pr(\rho_i \mid 1, |\rho_i|) &= \frac{f(\rho_i, |\rho_i| \mid 1)}{f(\rho_i \mid 1) + f(-\rho_i \mid 1)} \\ &= \frac{f(\rho_i \mid 1)}{f(\rho_i \mid 1) + f(\rho_i \mid 0)} \end{aligned} \tag{4.24}$$

and (4.23) and (4.24) are equal. We will denote this symmetric crossover probability simply as p_i.

Now consider a sequence of n outputs from a binary-input symmetric-output memoryless channel. Any sequence of n outputs belonging to

$$(\pm\rho_1 \quad \pm\rho_2 \quad \cdots \quad \pm\rho_n) \tag{4.25}$$

can be represented by the transition diagram shown in Figure 4.5, where the crossover probabilities are labeled by

$$(p_1 \quad p_2 \quad \cdots \quad p_n). \tag{4.26}$$

Larger conditional probabilities are shown as darker arrows, to emphasize that the channel reliability varies over the sequence of n bits.

A RT decoder views the outputs of Figure 4.5 as a reordered sequence. An example is shown in Figure 4.6, where the reordering puts the most reliable bits first. This corresponds to reordering the bits according to their crossover probabilities. In Figure 4.6 the ranked crossover probabilities are denoted as $p_{i:n}$, with the subscript $i:n$ indicating the i^{th} smallest order statistic from a sample of size n. A channel formed by reordering blocks of n bits into most-reliable-bit-first order will be referred to as an *Order Statistic Channel* (OSC).

On an OSC the crossover probability for any particular position will vary as different sequences of n outputs are observed. However, we propose to model each conditional transition probability by a single 'fixed' transition probability. A representative set of transition probabilities are taken to be the expected transition probabilities for each reordered position. The resulting model is referred to as the *Order Statistic Channel Model* (OSC Model), and is shown in Figure 4.7, where the 'fixed' transition probabilities are denoted as $\bar{p}_{i:n}$.

The justification for the OSC model stems from an asymptotic property of order statistics; specifically, that as n is increased, an OS is an asymptotic estimator of a quantile of the parent distribution [76][77]. Our direct interest is not in estimating a quantile, but rather we are interested in the property that the variance of an OS is reduced as n increases [76][77]. For RT decoding we are interested in modelling the effect of OS reordering on the transition probabilities $\{p_i\}_{i=1}^{n}$. As will be discussed, the effect of the OS reordering will be to decrease the variability of these transition probabilities. As n is increased, the varying transition probabilities for a reordered position will be increasingly well represented by the expected transition probabilities used to form the OSC model.

Let $F_X(x)$ denote the distribution of the r.v. X that is used to rank the symbol positions, with corresponding pdf $f_X(x)$. The n r.v.s $X_1, X_2, \ldots, X_n$ are assumed independent and identically

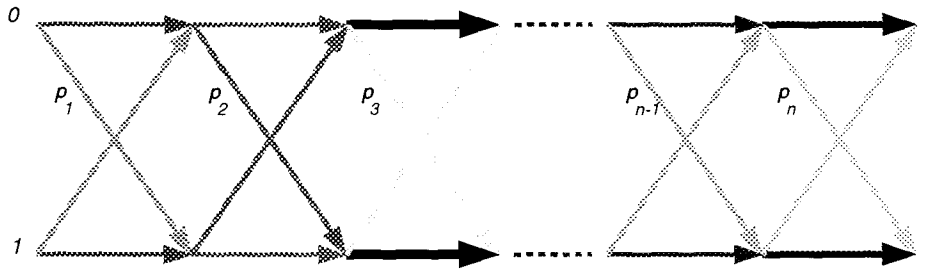


Figure 4.5 A Sequence of Symmetric Transition Probabilities
The reception of n outputs belonging to $(\pm\rho_1 \quad \pm\rho_2 \quad \cdots \quad \pm\rho_n)$ from a binary-input symmetric-output channel are represented here by symmetric transition probabilities. To emphasize that the bit reliabilities vary, larger transition probabilities are shown as darker arrows.

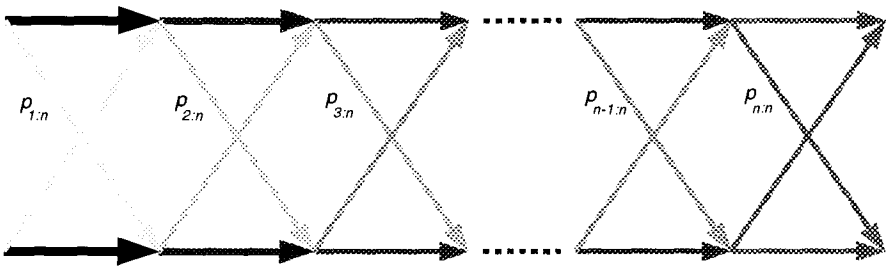


Figure 4.6 Reordered Transition Probabilities
The symbol position permutation used by RTD forms a channel consisting of reordered transition probabilities. Here the i^{th} crossover probability is labeled as $p_{i:n}$ to indicate that it is the i^{th} order statistic from a sample of size n. As in Figure 4.5 larger transition probabilities are shown with darker arrows, which displays the most reliable bit reordering.

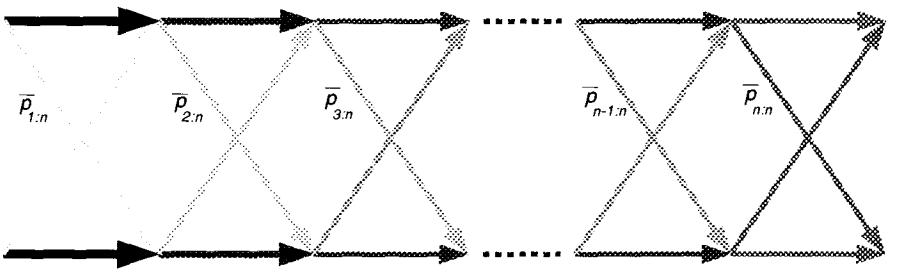


Figure 4.7 The Order Statistic Channel Model
The OSC Model represents the channel viewed by the RT decoder as having fixed transition probabilities, which are the expected transition probabilities for each reordered bit. The expected crossover probabilities are denoted as $\bar{p}_{1:n}, \bar{p}_{2:n}, \ldots, \bar{p}_{n:n}$.

distributed (i.i.d.), which is consistent with our assumption of a memoryless channel. A standard result [76] is that the pdf for the h^{th} smallest OS drawn from n samples of a parent i.i.d. r.v. with distribution $F_X(x)$ is

$$f_{X_{h:n}}(x) = h\binom{n}{h} F_X^{h-1}(x)[1 - F_X(x)]^{n-h} f_X(x) \qquad , 1 \leq h \leq n. \tag{4.27}$$

Following the approach used in [77] the pdf of the OS given by (4.27) can be written as the product of the pdf $f_X(x)$ of the parent distribution with another function $W_{h:n}(x)$ given by

$$W_{h:n}(x) = h\binom{n}{h} F_X^{h-1}(x)[1 - F_X(x)]^{n-h} \qquad , 1 \leq h \leq n. \tag{4.28}$$

To illustrate the 'behaviour' of the expected value of an OS we will first make a change of variables, $u = F_X(x)$, so that (4.28) becomes

$$w_{h:n}(u) = h\binom{n}{h} u^{h-1}[1 - u]^{n-h} \qquad , 1 \leq h \leq n, \;\; 0 \leq u \leq 1. \tag{4.29}$$

This function is referred to in [77] as the *sort function.* The expected value of an OS is then [77][76]

$$\begin{aligned} E[X_{h:n}] &= \int_{-\infty}^{\infty} x \, f_{X_{h:n}}(x) \; dx \\ &= \int_{-\infty}^{\infty} x \, W_{h:n}(x) \, f_X(x) \, dx \\ &= \int_{0}^{1} w_{h:n}(u) \, F_X^{(-1)}(u) \; du \, . \end{aligned} \tag{4.30}$$

where $F_X^{(-1)}(u)$ denotes the inverse function of $F_X(u)$. Equation (4.30) reveals why $w_{h:n}(u)$ is called the 'sort function', since an expected OS is the integral of a function ($F_X^{(-1)}(u)$) dependent solely on the parent distribution, weighted by the sort function, *which is independent of the parent distribution.* Figure 4.8 shows representative sort functions for increasing n. The area under the

graph of each sort function is unity, since the sort function is precisely the Beta pdf. As n is increased, we see that the expected value of an OS will be influenced by an increasingly narrower region of u, which implies that the region of influence of the parent distribution will also decrease. As n is increased, the sort functions approach impulse functions [77].

Each transition probability in the OSC model is the expected transition probability for a reordered position, where the expectation is over the order-statistic pdf of the ranking r.v. X. Let $p(x)$ denote the crossover probability as a function of x. Then the expected crossover probability for the h^{th} reordered position is

$$\begin{aligned} \bar{p}_{h:n} &= E_{X_{h:n}}[p(x)] \\ &= \int_{-\infty}^{\infty} p(x)\, W_{h:n}(x) f_X(x)\, dx\,. \end{aligned} \tag{4.31}$$

In terms of the variable $u = F_X(x)$, it is easy to show that (4.31) becomes

$$\bar{p}_{h:n} = \int_0^1 w_{h:n}(u)\, p\big(F_X^{-1}(u)\big)\, du, \tag{4.32}$$

so that the accuracy with which the OSC model transition probabilities represent the actual transition probabilities will increase with n, due to the asymptotic behaviour of the sort function $w_{h:n}(u)$.

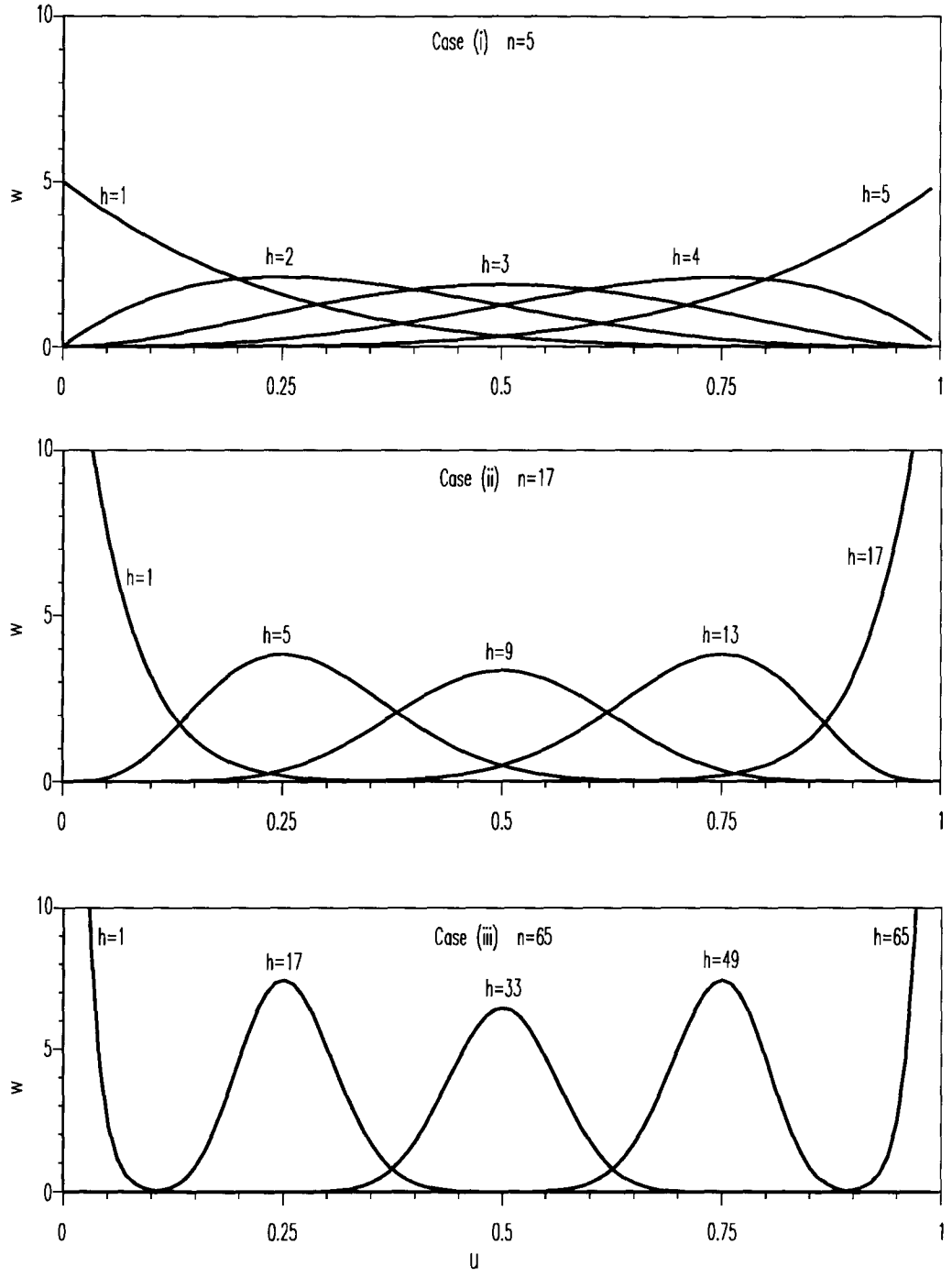


Figure 4.8 Examples of the Sort-Function for Different Sample Sizes

Some representative sort-functions are shown for $n = 5, 17, 65$ plotted against the value $u = F(x)$ of the parent distribution. Each sort-function has unit area. In Case (i) (n=5) the OS sort-functions $w_{h:n}(u)$ for h=1,2,...,5 are shown, with modal points at $u = \frac{h-1}{n-1} = 0, 0.25, 0.5, 0.75$, and 1. In the graphs for Cases (ii) (n=17) and (iii) (n=65), we have shown only those sort functions that correspond to these same modal points, for ease of comparison.

4.5.2 Approximate Pruning Impact of the RT-M Algorithm

Here we develop a method to estimate the pruning impact of the RT-M algorithm. The approach utilizes:

i. The uniform error property for pruning decoders. (Theorem 4.1)

ii. The OSC Model. (Section 4.5.1)

iii. The upper bound on the pruning impact. (Eqn. (4.22))

First we consider an upper bound on the pruning impact of the RT-M algorithm. In RT decoding the searches are carried out on trees or trellises corresponding to all of the codes $\mathbf{C}'$ equivalent to $\mathbf{C}$. The decoder knows with certainty which $\mathbf{C}'$ to decode for each trial since the decoder itself has specified the symbol position permutation. Let $P_I(\mathbf{C}')$ denote the pruning impact of the decoder for the code $\mathbf{C}'$. The pruning impact, considering all equivalent codes, is then

$$P_I = \sum_{\mathbf{C}'} Pr(\mathbf{C}')\, P_I(\mathbf{C}') \tag{4.33}$$

An upper bound on (4.33) is clearly

$$P_I \leq \max_{\mathbf{C}'}\ P_I(\mathbf{C}'). \tag{4.34}$$

We now argue that the systematic code $\mathbf{C}$ is the maximizing code in (4.34) for the M algorithm. We first compare the pruning impact of *tree* decoding a systematic code $\mathbf{C}$ versus that of a *particular* non-systematic code $\mathbf{C}'$. The particular code $\mathbf{C}'$ differs from $\mathbf{C}$ only in that it has a constraint-symbol at position k and an information-symbol at position $k+1$. Since the M algorithm will prune only at information-symbol positions, the number of heads pruned when decoding $\mathbf{C}$ at depths k and $k+1$ will be M' and 0, respectively[29]; while for $\mathbf{C}'$, there will be 0 and M' heads pruned

[29] The number of heads pruned is M', where $M' \leq M$. M' will equal M when there were M survivors that were extended to form $2M$ contenders (in tree decoding) of which M are pruned to leave M survivors. M' can be less than M due to merges when using trellis decoding, or at the early depths of the tree or trellis where the number of contenders can be significantly less than $2M$.

respectively. Decoding C will have the same or larger pruning impact than decoding C$'$, since in the former case the M algorithm is constrained to prune the same number of codewords but based on less head-likelihood information. In other words, when decoding C the M algorithm is more severely constrained in the head-likelihood information that can be utilized in deciding on which heads to prune. Now, any non-systematic code can be considered to be a systematic code that has been altered to have some constraint-positions occur earlier. Any such early constraint-positions postpone the pruning in the M algorithm (as described above for C$'$) and the consequence of this postponement is the accumulation of more head-likelihood information on which to base the pruning decisions. Finally, similar arguments hold for trellis decoding, where the early constraint-positions of a non-systematic code similarly postpone the pruning.

We now exploit the OSC model to estimate the pruning impact of the RT-M algorithm when decoding a systematic code. First, it is convenient to relabel the OSC model outputs ρ_i and $-\rho_i$ as 0 and 1 respectively. With this relabeling an output label j can be written as the modulo 2 sum of an input label k plus an error label e, i.e. $j = k + e$. For transmitting a codeword using the OSC model, we can then conveniently write the vector of output labels $\mathbf{y}$ as$\mathbf{y} = \mathbf{c} + \mathbf{e}$ where $\mathbf{c}$ is the codeword and $\mathbf{e} = (\, e_1 \quad e_2 \quad \cdots \quad e_n \,)$ is an error vector. From Theorem 4.1 we can then assess the pruning impact assuming that only the all zero codeword $\mathbf{c}_0$ has been transmitted, so that $\mathbf{y} = \mathbf{c}_0 + \mathbf{e} = \mathbf{e}$.

We now utilize the upper bound (4.22) on the pruning impact. For the OSC model it is appropriate to use a discrete version of (4.22), where we replace the continuous output vector $\mathbf{r}$ with the discrete output vector $\mathbf{y}$, to obtain

$$\widehat{P}_I \approx \sum_{E_0} Pr(\mathbf{e} \mid \mathbf{c}_0 \text{ transmitted}) \tag{4.35}$$

where E_0 denotes the set of error vectors for which $\mathbf{c}_0$ is pruned. We have used the symbol $\approx$ to remind us that this is an approximation to $\widehat{P}_I$ due to the use of the OSC model. We are interested

in evaluating (4.35) for the case of decoding a systematic code. To do this we will exploit the systematic form of the code together with the fact that the M algorithm will only prune in the information positions.

Using the OSC model we can draw a tree showing the head likelihood-distances when decoding a systematic code, as shown in Figure 4.9 for the RT-M algorithm with $M = 4$. Recall that since we are using the OSC model and transmitting $\mathbf{c}_0$, we have that the output is $\mathbf{y} = \mathbf{e}$. Consequently, we can write a head likelihood-distance $D(\mathbf{y}^h, \mathbf{c}^h)$ (see (3.27), page 67) as $D(\mathbf{e}^h, \mathbf{c}^h)$. Observe that for a systematic code the set of likelihood-distances for all heads of length $l \leq k$,

$$\left\{ D\left(\mathbf{e}^h, \mathbf{c}^h\right);\ \text{any } \mathbf{c}^h \text{ of length } l \leq k \right\} \tag{4.36}$$

will be invariant to $\mathbf{e}^h$, since any head $\mathbf{c}^h$ of length $l \leq k$ is allowed in a systematic code. Now recall that the M algorithm operating on a systematic code will only prune heads of length $l \leq k$, so that with the set of likelihood-distances for those heads being invariant to $\mathbf{e}^h$ we have that the set of head likelihood-distances that are pruned will also be invariant to $\mathbf{e}^h$. We will denote this set of likelihood-distances as D_P. In the example of Figure 4.9, D_P corresponds to the likelihood-distances marked with 'x's.

Consider that as $\mathbf{e}$ is varied, any specific member $D(\mathbf{e}^h, \mathbf{c}^h)$ of D_P may correspond to different codeword heads. We are interested in how often $\mathbf{c}_0$ gets pruned, and to compute this we will sum the contributions to (4.35) from pruning each $D(\mathbf{e}^h, \mathbf{c}_0^h)$ in D_P. Now, since we have transmitted $\mathbf{c}_0$, any particular $D(\mathbf{e}^h, \mathbf{c}_0^h)$ occurs with probability $Pr(\mathbf{e}^h \mid \mathbf{c}_0^h)$, since this is precisely the probability that $\mathbf{c}_0^h$ is transmitted and received as $\mathbf{e}^h$. As well, when an $\mathbf{e}^h$ is pruned, any tail $\mathbf{e}^t$ will also be pruned, so that the probability of $\mathbf{c}_0$ being pruned when $\mathbf{e}^h$ is pruned is

$$Pr\left(\mathbf{e}^h \mid \mathbf{c}_0^h\right) \sum_{\mathbf{e}^t} Pr(\mathbf{e}^t \mid \mathbf{c}_0^t) = Pr\left(\mathbf{e}^h \mid \mathbf{c}_0^h\right). \tag{4.37}$$

Eqn. (4.37) is the contribution to (4.35) for always pruning a $\mathbf{e}^h$ corresponding to a member of D_P. Considering all of the members of D_P, we obtain for (4.35)

$$\widehat{P}_I \approx \sum_{\mathbf{e}^h \leftarrow D_P} Pr\left(\mathbf{e}^h \mid \mathbf{c}_0^h\right). \tag{4.38}$$

Using (4.38) we can estimate the upper bound on the pruning impact (4.22) by a *single* M algorithm search of a depth k tree, based on the OSC model, as explained below.

The estimation procedure is perhaps easiest to explain by referring to Figure 4.9. Label each branch in Figure 4.9 with its corresponding OSC model transition probability as follows. Each upward branch corresponds to an error, so that each upward branch at depth i corresponds to the OSC model crossover probability $\bar{p}_{i:n}$. Similarly, each horizontal branch corresponds to a correct bit, and is labeled with $1 - \bar{p}_{i:n}$. To estimate the pruning impact, one extends and prunes heads in the systematic (unconstrained) tree using the M algorithm, and accumulates the probability of each head that is pruned (this probability is simply the product of the head's branch probabilities). The sum of the probabilities of the pruned heads is the sum of all of the probabilities of reaching the nodes marked X in Figure 4.9. This sum is precisely the estimate of the pruning impact (4.38).

The above estimation procedure is simple and efficient since:

i. It is independent of the detailed structure of the code.

ii. It requires only a relatively small number of nodes to be explored (since the exploration is limited by M, which is small in relation to the number of states of the code).

iii. It avoids considering an infinite set of outputs of a soft-decision channel (by using the OSC model, which has only 2 outputs per position).

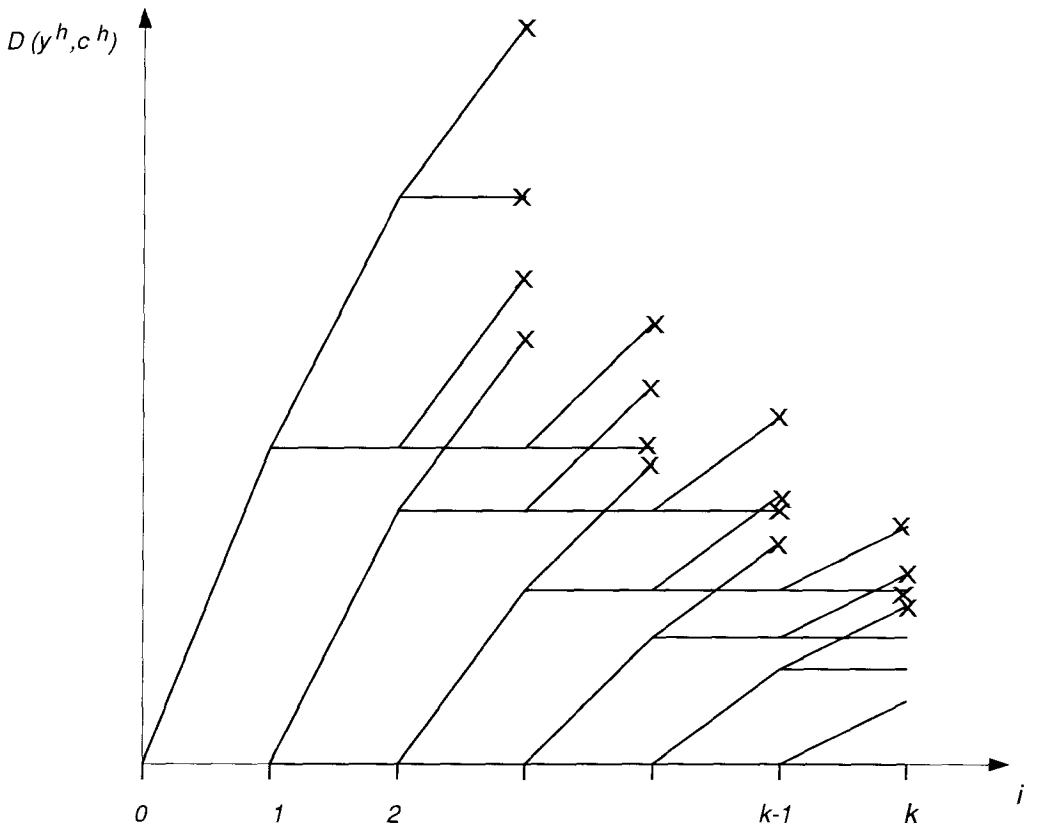


Figure 4.9 Head Likelihood-Distance versus Depth for the RT-M Algorithm with a Systematic Code. The explored head likelihood-distances are shown for a binary systematic code using an OSC Model with $M = 4$, for depths $0, 1, ..., k$. Pruned heads are indicated by X's.

4.5.3 Estimated WER of the RT-M Algorithm

Figure 4.10 compares the WER obtained via simulation with the WER estimated using the estimation procedure discussed above. The binary (24,12) Golay code is used on an AWGN channel with binary PSK. The curve labeled "RT-M" shows the simulation results for the WER of the RT-M algorithm for various M, and the error bars indicate 95% confidence intervals. The curve labeled "Estimated RT-M" uses (4.38) as discussed to estimate the pruning impact, and then forms an estimate for the WER by adding an upper bound on $P_{e,ML}$. The upper bound on $P_{e,ML}$ was taken to be the WER for M=64. It can be seen that the WER versus M curve is virtually flat for $M \geq 32$, so that the M=64 WER value should be reasonably close to $P_{e,ML}$.[30] The error bars from the simulation of M=64 are incorporated into the "Estimated RT-M" plot.

The estimated WER exceeds the simulation results by less than about 30% for a SNR of 2 dB. This excess corresponds to a small dB loss (< 0.25 dB), which can be estimated from Figure 4.3 (pg. 96). It can also be seen that this excess decreases for higher SNR. To account for this effect, let us review the sources of excess[31] in the estimate of the pruning impact, namely, the use of:

i the use of the systematic code to upper bound the pruning impact,

ii the upper bound on the pruning impact (Eqn. (4.22)).

For increasing SNR, the errors that occur in a block are increasingly confined to the poorest ranked bits. For example, the number of errors occurring outside of the $d_{min} - 1$ poorest ranked bits will decrease with increasing SNR. Consider the situation where the SNR has increased such that effectively all of the errors that occur are confined to the $d_{min} - 1$ poorest ranked bits. In this

[30] This can also be verified by comparison with the lower bound on the VA shown in Figure 4.3.

[31] The accuracy of the estimate is also affected by the use of the OSC model, which may contribute positively or negatively to the excess. We have previously discussed that this error diminishes as n is increased.

situation, decoding the systematic code will give effectively the same error probability as decoding any equivalent code, since both codes have the same distance properties for the positions where the errors do occur. This is due to the fact that all equivalent codes will have the last $d_{min} - 1$ bits as constraint bits. This accounts for a decreasing excess in the estimate of the pruning impact due to item (i) as the SNR is increased.

The increasing confinement of errors to the poorest ranked positions as the SNR is increased will also imply that the excess due to item (ii) will decrease. This can be seen as follows. Recall that the upper bound on the pruning impact is (4.16)

$$\widehat{P}_I = Pr(\text{CP pruned}) \tag{4.39}$$

and the excess of this upper bound is (4.17)

$$Pr(\text{ CP pruned } \cap \text{ CP} \neq \text{ ML path}). \tag{4.40}$$

However, with M fixed and the SNR increasing, the increasing confinement of the errors to the $d_{min} - 1$ poorest positions implies that the pruning decoder WER will approach that of ML decoding.[32] This implies that $Pr(\text{CP pruned})$ becomes small in relation to $P_{e,ML} = Pr(\text{CP} \neq \text{ ML path})$, which in turn implies that (4.40) decreases in relation to $P_{e,ML}$.

[32] Recall that if all errors were confined to constraint positions the decoder can attain ML decoding with only M=1. (See pg 94.)

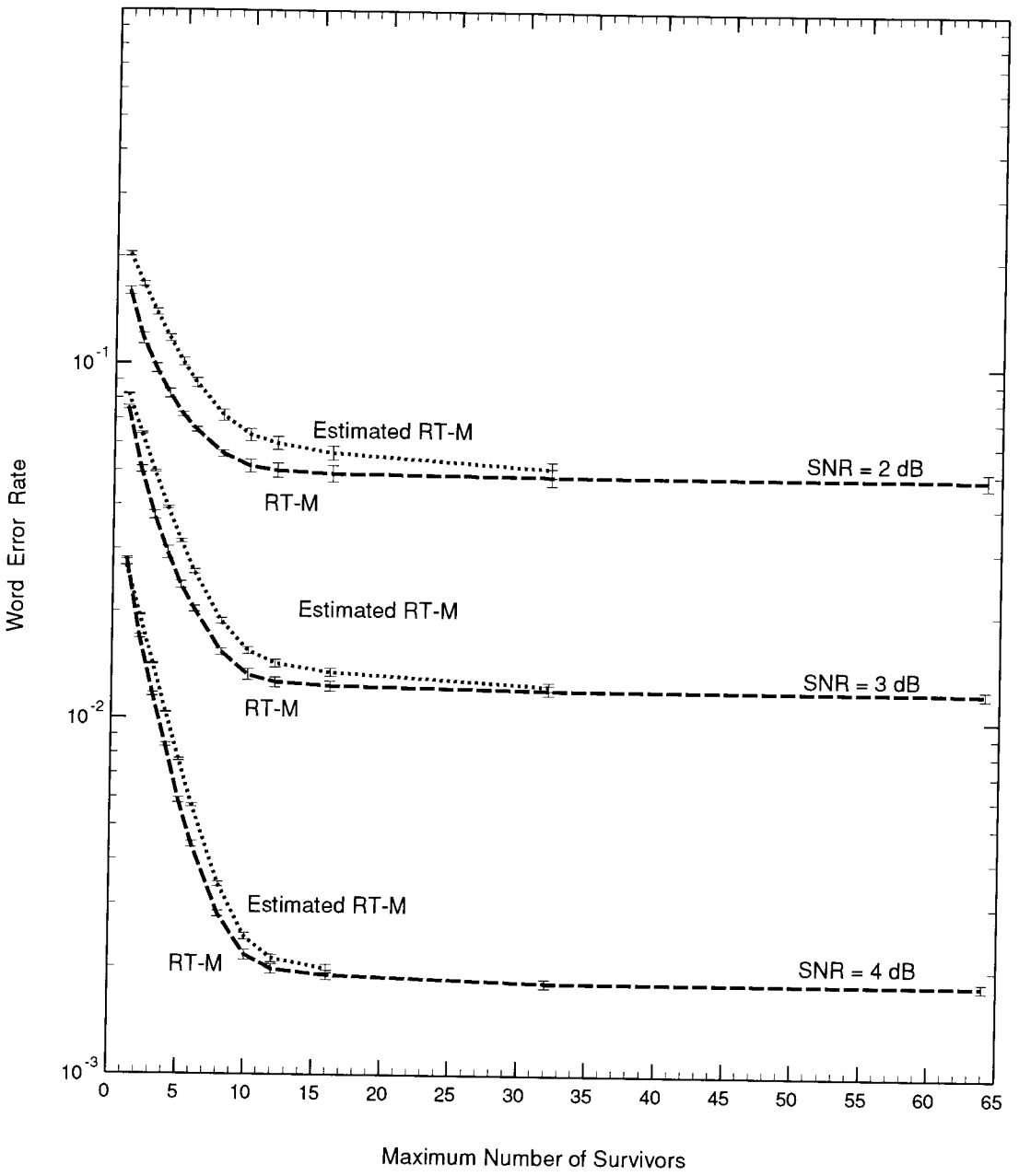


Figure 4.10 Word Error Rate of the RT-M Algorithm, Simulation and Estimation Results
The word error rate is plotted vs. the maximum number of survivors for the (24,12) Golay code on an AWGN channel with BPSK.

4.6 Computational Effort of RT-M Decoding

In discussing the computational effort of RT decoding we will assume that the contender merge-sifting (CMS) method of §3.4 is used to implement the M algorithm. We will make comparisons of this RT–M algorithm with both ML decoding and near-ML decoding. Specifically, RT-M is compared to:

i The most efficient soft-decision ML decoders for a specific code, namely the decoders of Vardy and Be'ery [78], Snyders and Be'ery [64], Forney [16], and Conway and Sloane [62] , for the binary extended Golay (24,12) code.

ii The M algorithm operating on various binary linear block codes.

As will be discussed, case (i) provides a comparison against highly specialized decoders, while case (ii) provides a comparison against decoding of general linear block codes.

4.6.1 The Number of Operations for RT-M Decoding

Here we consider the number of metric operations (such as additions and comparisons) and the non-metric operations (such as the binary-vector operations involved in carrying out partial searches) carried out by RT-M decoding. We assume that binary linear block codes are used.

Number of Metric Comparisons, N_c

In §3.4 upper bounds were found on the worst-case number of metric comparisons to implement the M algorithm using contender merge-sifting, for binary linear block codes. For RT-M decoding the decoder will be decoding all codes equivalent to the original transmitted code, so that an appropriate upper bound from §3.4 is (3.39). Equation (3.39) upper bounds the worst-case number of metric comparisons for the worst-case equivalent-code, which has its last information-symbol at

depth $n - d_{min}$. Equation (3.39) is

$$N_c \leq M(k + \Delta + 2 - l_M) + 2^{l_M} - l_M - 2 - \Delta \tag{4.41}$$

where $l_M = \lfloor \log_2(M) \rfloor + 1$ is the first depth where the number of contenders exceeds M, and $\Delta = n - k + 1 - d_{min}$ is the amount by which d_{min} is less than the Singleton bound.

As well, a looser but simpler upper bound is given by (3.41)

$$N_c < Mk + (M - 1)\Delta. \tag{4.42}$$

Number of Metric Additions, N_a

The worst case number of metric additions in the M algorithm implemented using CMS can be upper bounded as follows. For $k - l_M$ information-positions where M survivors are extended to form $2M$ contenders, there will be exactly M contenders in each of the contender-subsets. Assume, without loss of generality, that the contender-subset C_0 corresponds to the hard-decision, for every depth. Since the contender-subset C_0 has metrics that are identical to those of the previous survivors, no additions are required here. The M contender metrics in C_1 can be computed with M additions, for each of these $k - l_M$ depths. For each of the $n - k$ constraint-positions, the worst-case number of additions occurs when all of the M contenders fall in C_1, again using M additions. For the initial depths $1, 2, \ldots, l_M - 1$ of the tree before there are M contenders, we can upperbound the number of additions as M, at each of these depths. Consequently, a simple upper bound on the worst-case number of metric additions is M additions for each of the n depths, giving

$$N_a < Mn. \tag{4.43}$$

An upper bound on the expected number of additions can be obtained by more carefully accounting for the number of additions in the initial depths of the tree, and by upper bounding the expected number of additions at constraint positions. In the depths less than or equal to l_M, the number of additions is the sum of the number of entries in C_1, which is at most

$$\sum_{i=1}^{l_M} 2^{i-1} = 2^{l_M} - 1. \tag{4.44}$$

For each constraint position, we have found that the worst case number of additions is M, and the expected number of additions is $M/2$, as follows. First, recall that in a binary linear block code the number of '0' bits and '1' bits is identical for each symbol position, so that with equiprobable information-bits the parity bits will also be equally likely to be a '0' or a '1'. In other words, at a constraint-position the bit is just as likely to be a '0' as it is to be a '1'. As discussed earlier, extensions using all '0' bits or all '1' bits require zero and M additions, respectively — so that with the constraint bits being '0' or '1' with equal frequency we have that the expected number of additions to form M contenders at each constraint depth is then simply $M/2$. Using (4.44) for the number of additions in the initial l_M depths, plus M additions for the each of the remaining $k - l_M$ information positions (at most), plus M/2 expected additions for the $n - k$ constraint positions, we obtain

$$\begin{aligned} E[N_a] &\leq 2^{l_M} - 1 + M(k - l_M) + \frac{M}{2}(n - k) \\ &= M\left(\frac{n+k}{2} - l_M\right) + 2^{l_M} - 1 \,. \end{aligned} \tag{4.45}$$

Symbol Position Ranking

The ranking of the received symbols could be accomplished using a comparison-based sorting algorithm. For small values of n the associated computational effort may be acceptable. However,

much more efficient sorting schemes can be employed to exploit the nature of the distribution of the input sequence, as described below.

Recall that the n values to be sorted are assumed to be i.i.d. with common distribution $F_X(x)$. We can divide the domain of the distribution to form n quantization regions such that the probability of X falling in any region is $1/n$. This quantization transforms the continuous r.v. X into a uniformly distributed discrete r.v. Sorting of uniformly distributed inputs can be accomplished using a bucket-sort in $O(n)$ expected time [45]. The efficiency of the bucket-sort stems from splitting the sorting procedure into the quantization stage (which categorizes inputs into buckets), and a sorting stage that sorts each bucket's contents. The efficiency arises with uniformly distributed inputs since the expected number of entries per bucket is unity; hence few buckets will require sorting, with little sorting effort expected per bucket.

The sorting effort can be further reduced by using approximate sorting, in which we dispense with sorting the bucket contents. Such approximate sorting should be sufficient for RT decoding as n increases, for the following reason. Consider that the approximate sorting can be viewed as quantizing the decoding metric to n levels. Since it is well known that soft-decision decoding can be accomplished on AWGN channels with a very small dB loss using only a small number b of bits for quantization (3 bits are typically used) [69], one would expect that providing roughly 2^b buckets should be sufficient. For $n > 2^b$, the number of buckets will exceed the number of quantization levels known to achieve good soft-decision decoding performance, so that approximate bucket sorting using n buckets should be sufficient. For $n >> 2^b$ it should then also be sufficient to have fewer than n buckets. With only a few bits being devoted to the metric, the metrics can be assigned to their appropriate buckets in worst case time of precisely n steps (and using zero comparisons) using the binary metric values as a direct address to index the array of buckets.

Number of Binary Vector Operations for Code Matrix Reduction, N_r

Here we consider the number of binary vector operations to reduce the parity check matrix **H** or the generator matrix **G**, for use with the 'Method 2' or 'Method 3' trellis generation schemes, respectively, discussed in Chapter 2.

Using elementary row operations the reduction of the $(n-k) \times n$ matrix H can be done using of the order of $(n-k)^2 n$ bit operations. Since these bit operations are simple bit-by-bit XOR or zero-testing operations, they are ideally suited to implementation in digital hardware. As well, it is natural to process the matrix using operations on n–bit vectors, rather than bit by bit. The key operations in the matrix reduction are (1) the identification of the rightmost 1 to find a pivot from the remaining non-pivot rows, and (2) the addition of this row to other rows that share a 1 in this pivot position. We assume that we have a length-n binary-vector processing-block that can find the rightmost 1 in a row. We also assume that we have a length-n binary-vector processing-block that performs the vector-XOR on another row if it has a one in the pivot position. The worst case number of such binary-vector operations is obtained as follows. At the first step in the reduction there will be at most $n-k$ tests to find the rightmost 1 as the first pivot. This is followed by $n-k-1$ operations of the XOR block to eliminate 1's in the non-pivot rows. At subsequent steps, there are at most $n-k-1, n-k-2, \ldots, 2, 1$ tests to find pivots, with $n-k-1$ XOR-block operations for each pivot row. In total then there are

$$\sum_{i=1}^{n-k} i = \frac{1}{2}(n-k)(n-k+1) \tag{4.46}$$

tests for pivots and

$$(n-k)(n-k-1) \tag{4.47}$$

XOR-block operations. Hence the total number of length-n binary vector operations is upper-bounded by

$$\begin{aligned} N_r &\leq \frac{1}{2}(n-k)(n-k+1) + (n-k)(n-k-1) \\ &= \frac{3}{2}(n-k)(n-k-1/3) \\ &< \frac{3}{2}(n-k)^2 \, . \end{aligned} \tag{4.48}$$

The number of length-n binary vector operations for reducing the generator matrix $\mathbf{G}$ instead of $\mathbf{H}$ can be similarly obtained as

$$N_r \leq \frac{3}{2}\, k^2. \tag{4.49}$$

For codes with rate $R \geq 1/2$, which is typically the case, it is simpler to reduce $\mathbf{H}$ for use in Method 2, rather than reduce $\mathbf{G}$ and use Method 3.

Number of Binary-Vector Operations for Tree Searching, N_s

Here we account for the number of binary-vector operations used in carrying out the partial tree search. The search can utilize states, as in the trellis generation schemes of Methods 2 and 3, or the search can dispense with maintaining states, as will be discussed shortly.

First we consider the number of binary-vector operations for the Method 2 and Method 3 trellis generation schemes. Both of these methods will have an identical number of binary-vector operations, except for the reduction of the code matrix. As discussed in the previous section, typical codes have a rate $R > 1/2$ and will require fewer operations for matrix reduction using Method 2. Below we consider the worst case and expected number of binary vector operations.

In Method 2, for extensions using the zero bit the new state is unchanged, so that no vector-XOR operation is required. We thus need only count the extensions using a '1'. However, not

all extensions using a one need be counted, for two reasons. First, recall from the 'Method 2' trellis construction method that extensions at constraint-positions need not update the state, since the extended state will automatically have a zero in a specific bit. (This bit is indicated by a counter that starts at 0 and is incremented for each constraint position, so that at each constraint position the leading remaining bit of each state is set to zero.) Hence the decoder can simply ignore this bit for all states at the constraint-depth (and all subsequent depths). Second, of the head extensions using a '1' at each information-position, the decoder need only compute the new state for those contenders that are retained by contender merge-sifting. However, we will ignore this saving and will form a worst case upper bound by counting the number of extensions using a '1' at information positions. For the initial l_M depths the decoder will perform

$$\sum_{i=1}^{l_M} 2^{i-1} = 2^{l_M} - 1 \quad . \tag{4.50}$$

vector-XOR operations to update the contender states. For each of the remaining $k-l_M$ information-positions, there are M extensions using a '1' (although not all of these extensions will be kept). Hence, the number of binary-vector operations is upper bounded by

$$N_s < M(k - l_M) + 2^{l_M} - 1. \tag{4.51}$$

A simpler upper bound is given by counting M state extensions at each of the information positions, giving

$$N_s < Mk. \tag{4.52}$$

An upper bound on the expected number of state computations can be found by reconsidering the above worst-case calculations at information positions greater or equal to l_M. Consider that with equally likely information-bits, and a symmetric channel, we should not favour the hard-decision

being a '1' over it being a '0'. Thus, for depth l_M and later, it is equally likely that a case of all of C_1 being retained is just as likely as a case of all of C_0 being retained. In other words, it is equally likely that M or 0 states are computed. Similarly, other numbers of contenders being retained from the contender-subsets (e.g. $(M/2, M/2)$ or $(1, M-1)$ etc.) occur without bias towards either a '0' or a '1'. Hence the average number of state computations at an information-bit is $M/2$. We then have

$$E[N_s] \leq \frac{M}{2}(k - l_M) + 2^{l_M} - 1 \; . \tag{4.53}$$

Finally, we consider an alternate method of carrying out the partial search, that dispenses with calculating states. In the method discussed above, we have generated a partial tree by simply ignoring merges in a partial trellis. We used the states only as a means of keeping track of the encoding of heads. One can avoid retaining state information for each survivor (and instead retain just the head symbols) by using 're-encoding,' as follows. At an information-position a head is extended with all symbols of the code alphabet. At a constraint-position, in order that the head be part of a codeword the extended symbol must satisfy one of the parity-check equations (due to the fact that the parity check matrix has been reduced to row echelon form). The appropriate parity check equation is simply the i_P^{th} row of the reduced $\mathbf{H}$ matrix, where i_P is the constraint-position counter introduced earlier. Thus, each head is extended and 're-encoded' by ensuring that each constraint symbol satisfies its appropriate parity check equation.

An upper bound on the worst case number of binary-vector operations used in re-encoding can be formed by assuming that at each constraint-position we have M contenders, so that there are at most $M(n-k)$ binary-vector operations, since as each head is extended there will be $n-k$ constraints to be met.

This approach can also be used with the reduced **G** matrix, where we simply re-encode each parity symbol as required, using the appropriate column of **G**. This will also result in a worst case upper bound of $M(n-k)$ binary vector operations.

This 'no-state' (or re-encoding) approach can be roughly compared to the state approach of Methods 2 and 3, as follows. Methods 2 and 3 were shown to have a number of binary-vector operations upper bounded by Mk, whereas the re-encoding approach has a worst case upper bound of $M(n-k)$. Hence, re-encoding without maintaining states will be advantageous for high-rate codes.

Summary of Upper Bounds on the Number of Operations

Tables 4.1 and 4.2 summarize the most useful of the upper bounds found above on the worst case and expected number of operations. From the detailed bounds in the table we see that each of N_c, N_a and N_s are loosely upper-bounded by Mn. Tables 4.1 and 4.2 will be used in the following sections.

	Upper Bound on Worst Case	Upper Bound on Expected Case
N_c Comparisons	$M(k+\Delta+2-l_M)+2^{l_M}-l_M-2-\Delta$ (4.41) $(< Mk+(M-1)\Delta)$ (4.42)	
N_a Additions	Mn (4.43)	$M\left(\frac{n+k}{2}-l_M\right)+2^{l_M}-1$ (4.45)

Table 4.1 Upper Bounds on the Number of Metric Operations
The upper bounds in the table are for decoding any binary linear block code using the M algorithm. Contender merge-sifting (§3.4) is assumed to be used to implement the M algorithm.

		Upper Bound on Worst Case	Upper Bound on Expected Case
N_r Matrix Reduction	**H**	$\frac{3}{2}(n-k)^2$ (4.48)	
	G	$\frac{3}{2}k^2$ (4.49)	
N_s Searching	States	$M(k-l_M)+2^{l_M}-1$ (4.51) $(<Mk)$ (4.52)	$\frac{M}{2}(k-l_M)+2^{l_M}-1$ (4.53)
	No-states	$M(n-k)$ (4.51)	

Table 4.2 Upper Bounds on the Number of Binary-Vector Operations
The upper bounds in the table are for decoding any binary linear block code using the M algorithm.

4.6.2 Comparison to ML Decoding

One of the strengths of RT decoding is that it is applicable to general codes; it does not depend on the detailed structure of the particular code to be efficient. It may be argued that this is a weakness rather than an advantage, in that additional savings could be obtained by exploiting the detailed structure of the specific code. However, we will demonstrate that even with ignoring the detailed code structure, the RT-M algorithm can compare favourably against decoders that are highly specialized.

Here we have chosen to compare RT-M decoding to several ML decoding methods for the extended binary Golay (24,12) code. This particular code was chosen since decoders for the Golay (24,12) code have been one of the most extensively investigated, so that the decoding efficiency has been refined via many approaches. Specifically, we consider here the decoders of Vardy and Be'ery [78], Snyders and Be'ery [64], Forney [16], and Conway and Sloane [62].

Table 4.3 summarizes the number of metric-pair operations for various decoders, where a metric-pair operation is an addition, subtraction, or comparison of two metrics. Comparing decoders on the basis of such metric operations is the convention used for all of these decoders [78][64][16][62]. Non metric–pair operations, such as absolute value operations, data movement, memory lookup, and control operations are not included for any decoder in Table 4.3. The number of metric operations for the RT-M decoder was found using the expressions in Table 4.1, and taking the value of M to be 10. This value for M was chosen since the it attains a WER within 0.1 dB of ML decoding, as can be seen from Figure 4.3 (pg 96). The complete set of parameters used in the calculations of the upper bounds are $M = 10$, which implies that $l_M = 4$, and $(n, k, d_{min}) = (24, 12, 8)$, which implies that $\Delta = 5$. The RT-M decoder then uses at most $N_c = 155$ comparisons and at most $N_a = 240$ additions (with 155 additions as an upper bound on the expected number of additions),

for a total worst case upper bound of 395 metric operations. From Table 4.3 we see that this number of metric operations is the lowest of the decoders. This reduction in the number of metric operations is significant, especially given that we are comparing the worst case number of metric operations of a general-purpose decoding method to that of very refined special-purpose decoders for a highly structured code.

	RT-M Decoding (4.41) + (4.43)	Vardy & Be'ery [78]	Snyders & Be'ery [64]	Forney [16]	Conway & Sloane [62]
Metric-Pair Operations	395	695	827	1353	1584

Table 4.3 Metric-Pair Operation Counts for Decoding the Extended Binary Golay (24,12) Code
For RT-M decoding the value of M used was taken to be 10, which corresponds to an estimated 0.1 dB loss with respect to the ML decoders at a BER of 10^{-3}. Non-metric-pair operation counts are not included here for any decoder. For the RT-M decoder there are 311 binary-vector operations.

For completeness we also present the number of binary-vector operations used by RT-M decoding in reducing the parity check matrix and performing state calculations used in the partial tree search. For M=10, the expressions in Table 4.2 give a total of 311 binary-vector operations as a worst case upper bound. These consist of $N_H = 216$ operations to reduce the parity check matrix, and $N_s = 95$ state calculations. The upper bound on the expected number of state calculations is 55. These binary-vector operation counts are not easily compared to the ML decoders of this

section. In the next section we consider some pruning decoders that enable us to more easily compare the operation counts.

4.6.3 Comparison to M Algorithm Decoding

In this section we compare RT-M decoding to M algorithm decoding. This enables us to make a more direct comparison between the decoders, since they are both partial tree searches that share the same types of operations. Additionally, the comparison between these two decoders is perhaps more meaningful than the comparison of the previous section, since here both decoders are for *general* linear block codes.

Since we are interested in the trade-off of computational effort versus coding gain we have chosen to present the simulation results as plots of complexity versus coding gain. This provides a more direct comparison than the traditional 'waterfall' plots of BER (or WER) versus SNR, with M as a parameter. For each code we give a pair of plots, as in Figure 4.11 for the BCH (32,16,8) code.

The first pair of the plots in each Figure, for example Figure 4.11(a), shows M versus the coding gain at a BER of 10^{-3}. As discussed earlier, the number of metric comparisons and additions are both upper bounded by Mn, so that plotting M versus the coding gain gives a good indication of the relative number of metric comparisons or additions. The coding gain, which is the decrease in SNR relative to the uncoded case to attain a target BER, was estimated as follows. The SNR required to reach the target BER was calculated using the following expression for the BER of binary PSK on an AWGN channel,

$$p_{\text{uncoded}} = Q\left(\sqrt{2E_b/N_0}\right). \tag{4.54}$$

The simulation results for the BER versus SNR were interpolated to find the SNR at which a M algorithm decoder would meet the target BER. The difference between these two SNRs gives an

estimate of the coding gain. Also shown is the coding gain estimated using a single term union bound on the ML decoder BER

$$p_{\text{coded}} \lesssim \frac{d_{min}}{n} A_{d_{min}} Q\left(\sqrt{2\frac{E_b}{N_0} R\, d_{min}}\right), \tag{4.55}$$

where $A_{d_{min}}$ is the number of codewords of weight d_{min}. Equation (4.55) is based on a single term union bound on the WER [79]

$$P_e \lesssim A_{d_{min}} P_{e2}(d_{min}) \tag{4.56}$$

where $P_{e2}(d_{min})$ is the probability of error for two codewords of Hamming distance d_{min} apart. Equation (4.56) assumes that the only significant error causing events are those at distance d_{min}. This single-term union bound is more convenient than a multi-term union bound, since we only need to know $A_{d_{min}}$. Moreover, since it provides a higher estimate of the ML coding gain, this will indicate a more conservative value of M that would be required to approximately attain the ML coding gain. The BER p_{coded} is estimated from the WER P_e by assuming that the fraction of information bits affected is d_{min}/n on average. This approach to estimating the coding gain is much more accurate for typical SNRs than using the asymptotic coding gain, which from (4.54) and (4.55) is $10\log_{10}(R\, d_{min})$. For example, for the (32,16,8) code, the asymptotic coding gain is 6 dB, while the coding gain (at 10^{-3} BER) estimated using (4.54) and (4.55) is slightly more than 3 dB.

The second pair of the plots in each Figure, for example Figure 4.11(b), shows the total number of binary-vector operations (normalized by n) versus the coding gain. In all cases we have assumed that 'no–state' re-encoding is used to perform the tree search, so that $N_s \leq M(n-k)$ and $N_r \leq 1.5(n-k)^2$. For the M algorithm, the total number of binary-vector operations is just N_s. For the RT-M algorithm the total number of binary-vector operation is $N_s + N_r$.

In Figure 4.11 for the (32,16,8) code it can be seen that both the number of metric operations and the number of binary-vector operations for the RT-M decoder are significantly less than the

M algorithm. The RT-M algorithm rapidly approaches the estimate of the VA coding gain given by the single term union bound. For example, to be within 0.25 dB of the estimated ML coding gain, M=10 is sufficient. The rate of convergence to the estimated ML coding gain is considerably slower for the M algorithm.

In 4.11(b), where the number of binary-vector operations is shown, it can be seen that while the effort to reduce the parity check matrix forms a significant fraction of the total number of binary-vector operations for the RT-M algorithm, the total number of binary-vector operations is still significantly below that of the M algorithm for coding gains of interest.

Similar results are shown for other codes in Figures 4.12 and 4.13. In Figures 4.12 and 4.13 the metric operations and binary-vector operations are shown for the binary (32,21,6) extended BCH code and the binary (64,51,6) extended BCH codes, respectively.

Decoding Examples on a Rayleigh Fading Channel

Soft-decision decoding is particularly attractive in mobile and portable radio applications, where multipath propagation can severely degrade the uncoded BER performance. The propagation is frequently modeled as having no direct line-of-sight component, with the resulting amplitude distribution of the signal being well modeled by a Rayleigh distribution [7][80]. In such cases the coding gain can be much larger than for the AWGN channel [6]. Figures 4.14–4.16 show curves similar to Figures 4.11–4.13 for the same codes but assuming that a Rayleigh faded AWGN channel is used with binary non-coherent frequency-shift-keying (NCFSK). Independent bit errors are assumed[33], so that the channel appears memoryless. The uncoded BER is

$$p_{uncoded} = \frac{1}{2 + E_b/N_0}. \tag{4.57}$$

[33] For example, independence among bit errors can be obtained via interleaving (time diversity) or frequency diversity.

The receiver is assumed to have available the SNR during each bit (and that it is constant over each bit duration). This constitutes 'side-information' [Chase72] in addition to the hard-decision outputs of the binary NCFSK demodulator. A single term union bound on the coded BER is

$$p_{coded} \lesssim \frac{d_{min}}{n} A_{d_{min}} P_e\left(R\frac{E_b}{N_0}, d_{min}\right) \tag{4.58}$$

where $P_e(\gamma_0, d_{min})$ is the probability of bit error for d_{min} order diversity, with the average SNR on the channel being γ_0. An approximate expression for the error probability of such a diversity system is given by [81, Eqn. 21].

An example of the large coding gain attainable on such channels is shown in Figure 4.14 for the binary extended (32,16) BCH code. The estimated ML coding gain for a target BER of 10^{-3} found using (4.57) and (4.58) is approximately 16 dB. This coding gain increases for lower target BERS [6]. It can be seen from Figure 4.14 that the RT-M algorithm rapidly approaches this coding gain, with M=8 being near the ML coding gain. Similar results can be seen in Figures 4.15 and 4.16 for the (32,21) and (64,51) codes, respectively.

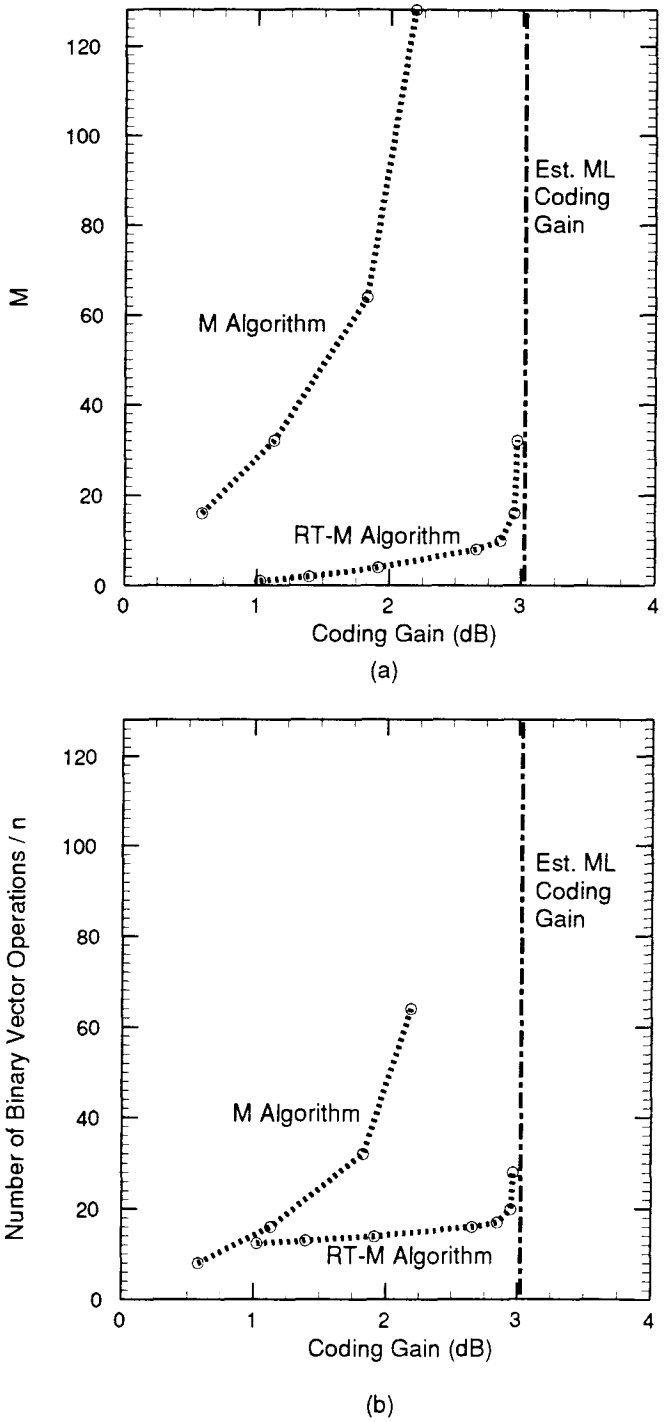


Figure 4.11 Number of Operations vs. Coding Gain : (32,16,8) Code on an AWGN Channel
In (a) the value of M is plotted versus the coding gain for a target BER of 10^{-3}. In (b) the number of binary comparisons normalized by the code length n is plotted against the coding gain. In both plots the vertical line shows the coding gain of ML decoding estimated from a single term union bound.

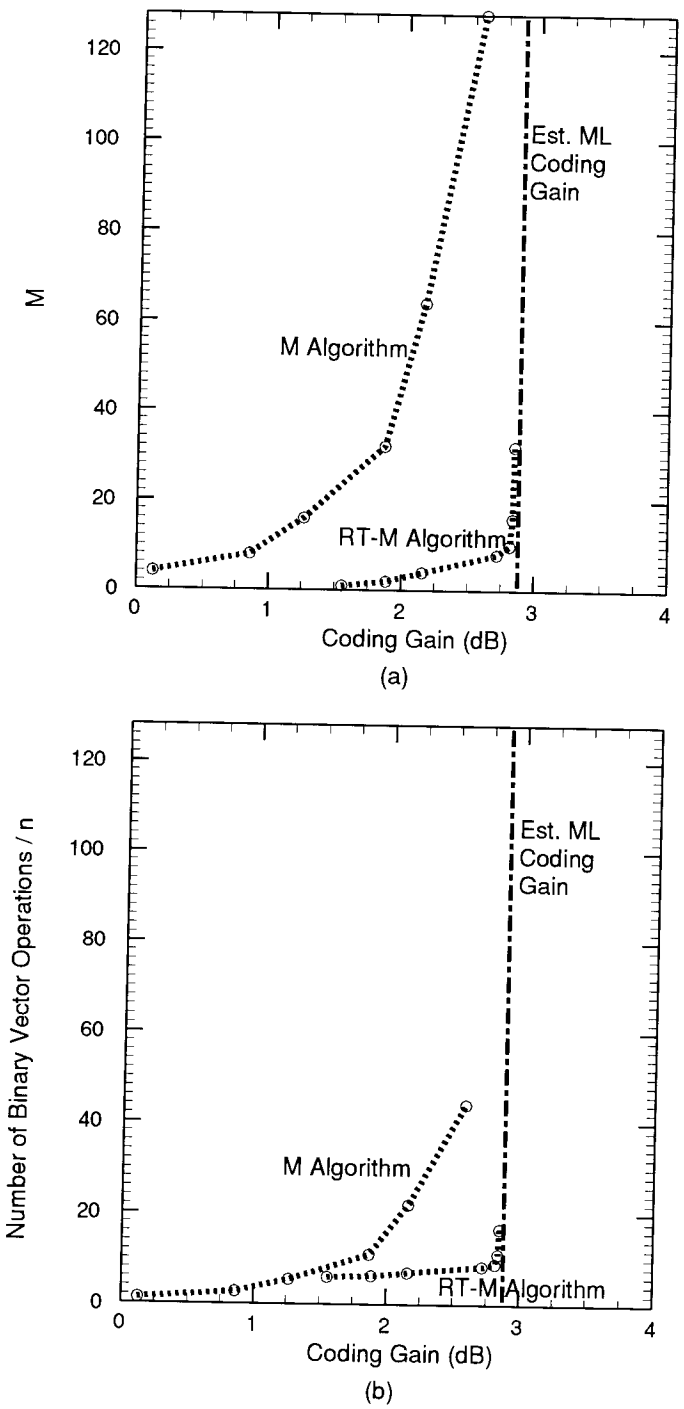


Figure 4.12 Number of Operations vs. Coding Gain : (32,21,6) Code on an AWGN Channel

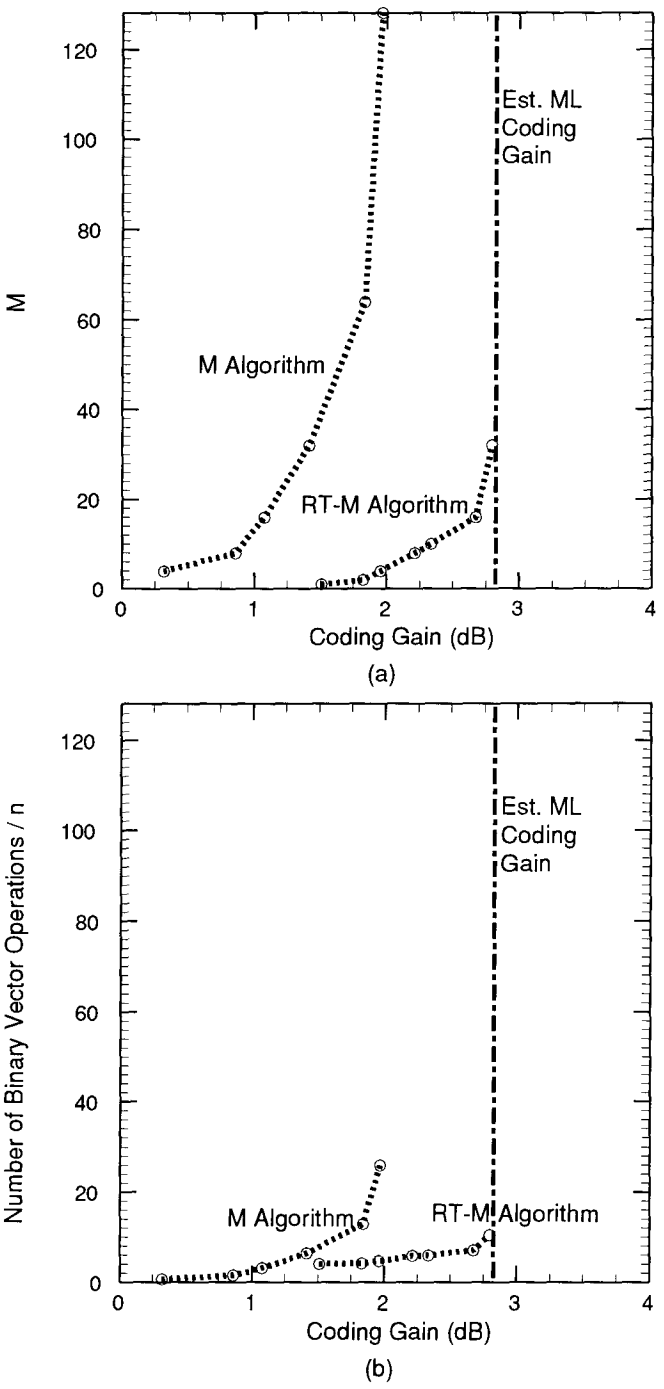


Figure 4.13 Number of Operations vs. Coding Gain : (64,51,6) Code on an AWGN Channel

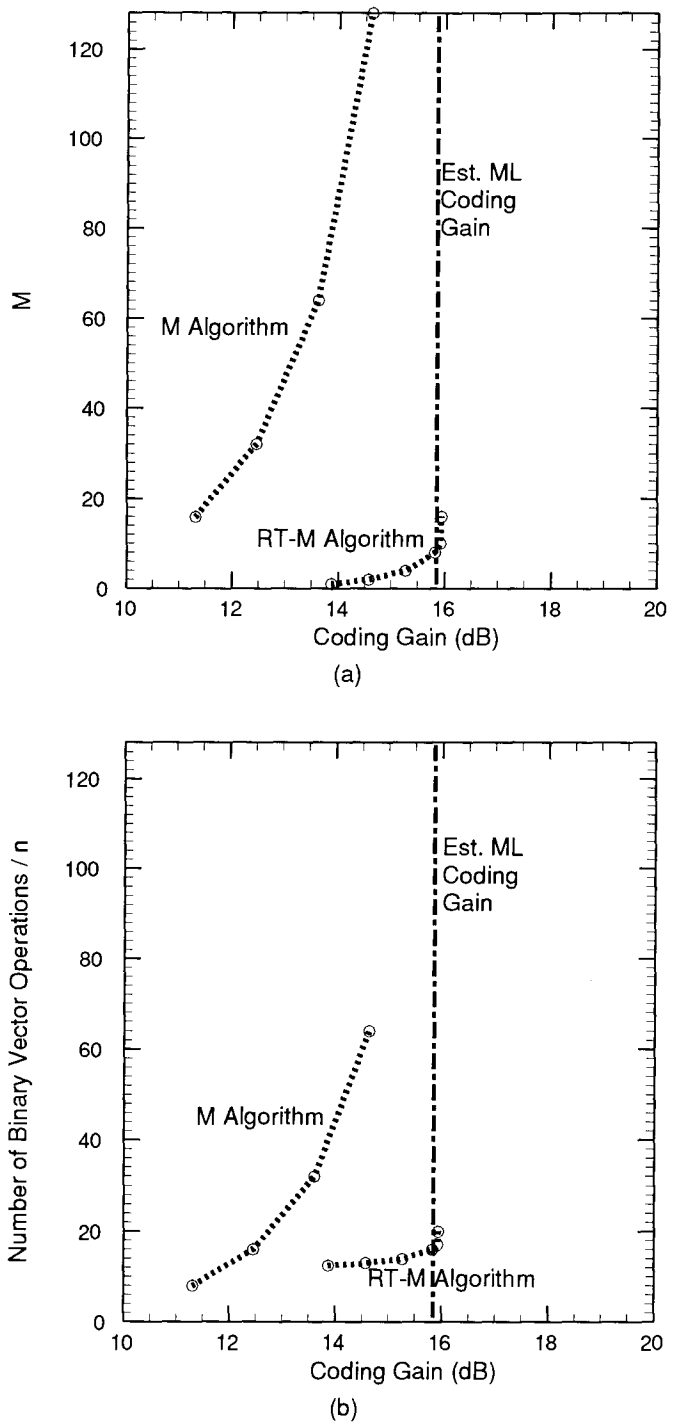


Figure 4.14 Number of Operations vs. Coding Gain : (32,16,8) Code on a Rayleigh Faded Channel
The channel has a Rayleigh faded signal amplitude, with AWGN. The signalling is binary NCFSK with the decoder using hard-decisions with 'side-information' consisting of the faded signal amplitude. In both plots the vertical line shows the coding gain of ML decoding estimated from a single term union bound.

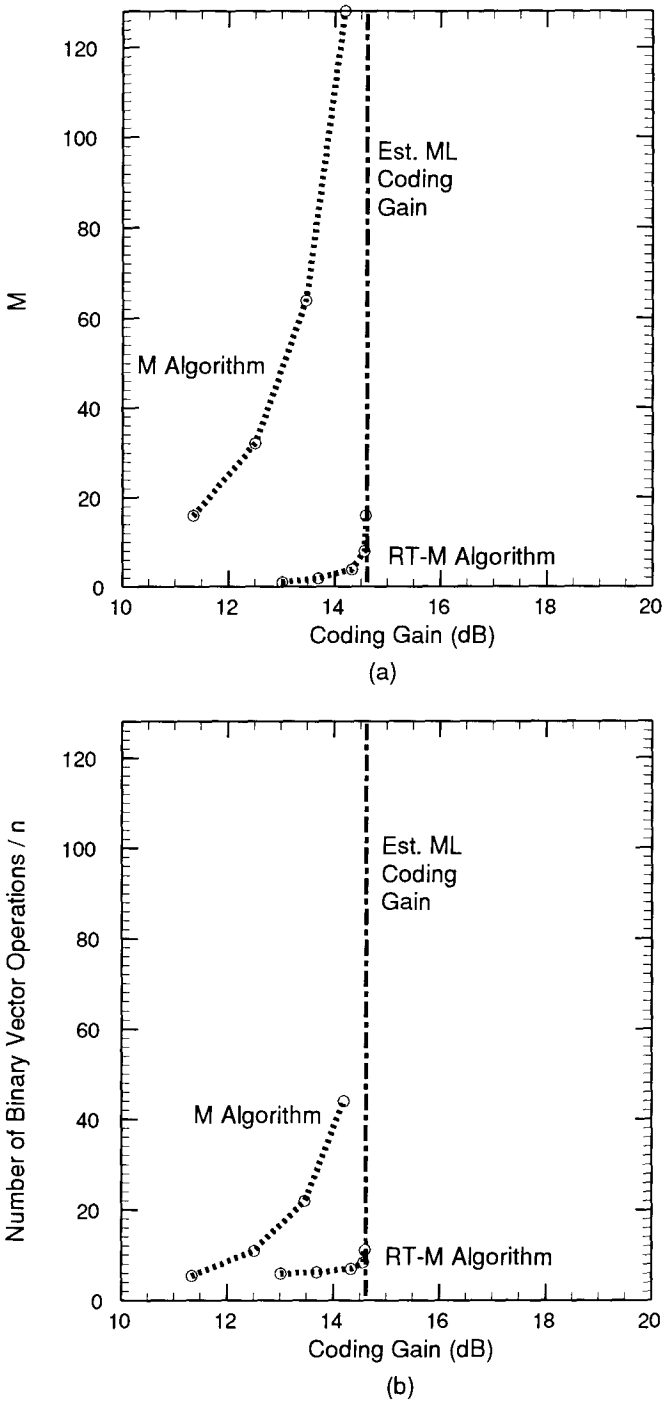


Figure 4.15 Number of Operations vs. Coding Gain : (32,21,6) Code on a Rayleigh Faded Channel

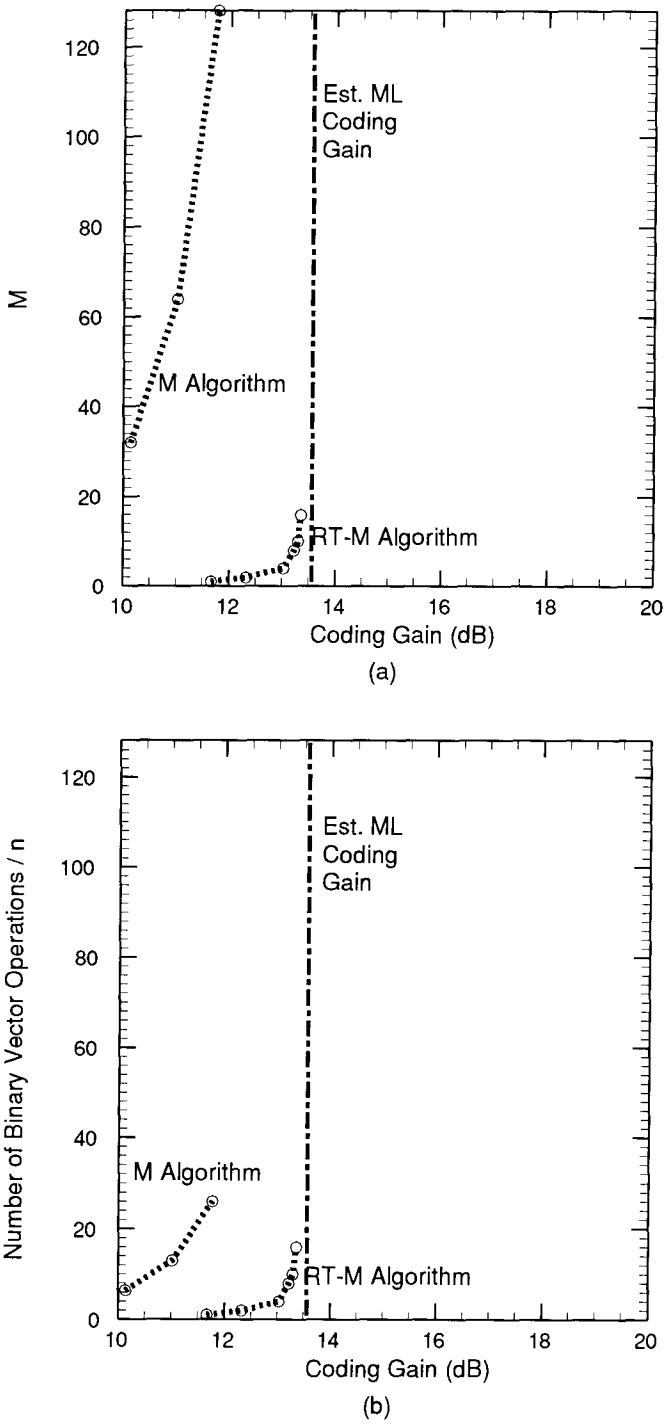


Figure 4.16 Number of Operations vs. Coding Gain : (64,51,6) Code on a Rayleigh Faded Channel

4.7 Discussion

The assessment of the computational effort of the RT-M algorithm in the previous section is intended to provide only a rough comparison against some other approaches. While we have endeavored to give a reasonably complete assessment of the computational effort, there are many other factors that influence the suitability of an algorithm and its implementation for a particular application. Despite this caveat, RT-M decoding appears attractive for decoding *general* linear block codes. Such versatility contributes greatly to the practical merits of RT decoding, since it may be feasible for a single implementation to be utilized in several coding applications — thus enabling improved economies-of-scale compared to specialized decoders.

It is useful to discuss the efficiency of RT decoding in terms of its resilience to error bursts. As discussed in [36] and elsewhere, such bursts are the bane of 'standard' sequential decoding. In either RT or standard decoding the decoder attempts to retain a sufficient number of survivors in order to keep the correct path (CP). In standard decoding, when the channel conditions are so poor as to offer little distinction between contenders, the decoder must retain a number of survivors that grows exponentially with depth. This exponential growth of the number of survivors becomes clear when we consider breadth-first tree decoding of a binary linear block code on an erasure channel. (This example is similar to one in [36] for sequential decoding of convolutional codes.) Assume that the first L bits of the codeword are received as erasures. Since the erasures reveal no information about which contenders are more likely to be the correct path, then even if as many as one half of the 2^L contenders that descend from the root node are kept in the breadth-first search, the decoder will only have retained the correct path with probability 1/2.

In contrast, the RT decoder need retain only *one* survivor over the first L bits to retain the CP with probability one. However, it may appear that the decoder has only postponed the inevitable,

since it must eventually process the erased bits. Note that all of the erased bits will have been reordered to be in the final L positions. If the number of erasures is within the minimum distance of the code, i.e. $L \leq d_{min} - 1$, then the RT decoder can deftly avoid retaining any more survivors, since the final $d_{min} - 1$ positions of the tree will be constraint positions, and as such they have only one branch descending from each node. The price paid for this more favourable search order is that the tree or trellis must be reconfigured to correspond to the reordered code. RT decoding will be efficient if the search savings are greater than the fixed computational 'overhead' of the code matrix reduction. Since the overhead to reduce the code matrix is of order $\min\left(k^2, (n-k)^2\right)$ binary-vector operations, the total number of operations in RT decoding can compare favourably against the exponential number of survivors for standard decoding.

While the preceding example is illuminating, the erasure channel does not fairly represent typical soft-decision channels. However, for such channels the action of RT decoding is similar to the erasure channel case. The RT decoder aims to avoid the need to retain a large number of survivors by reordering the bits so as to postpone the processing of unreliable bits. This allows the decoder to search through the initial depths of a code tree without processing unreliable bits — thus diminishing the number of survivors while retaining the CP with high probability. By the time the decoder gets to the less reliable depths of the tree there may be a sufficient number of constraints imposed by the code so that many errors are 'trapped'. In the examples presented in Section 4.6, the decrease in search effort offered by RT decoding was shown to greatly outweigh the increased computational overhead.

Some other factors contribute to the efficiency of RT decoding. First, the partial trellis exploration is facilitated by the use of the simplified trellis construction methods discussed in Chapter 2. Or, for tree decoding, we can simply re-encode the codewords as the search progresses.

Second, the efficiency of the RT-M algorithm is enhanced by the use of the Contender Merge Sifting (CMS) method introduced in Chapter 3. With CMS, the computation associated with sifting out the best survivors at each decoding stage is greatly reduced compared to other comparison-based sifting methods.

The efficiency of RT-decoding could be improved somewhat further by the following steps. The decoder could first check if the hard-decision vector has a zero syndrome, and only carry out decoding if the syndrome is nonzero. On high SNR channels this would lower the average decoding effort. Another way to lower the decoding effort of the RT-M algorithm is to extend only the hard-decision branches for some number of the first and most reliable bits, and/or to gradually increase the number of survivors allowed at each depth. This 'variable-M' approach is natural in the sense that it better matches the number of survivors to the order statistic channel. We have not reported on such an approach, since it will not reduce the peak number of survivors required in a breadth-first search.

Increased decoding delay will usually be necessitated in using RT decoding, due to the need to receive all of the bits in a block before beginning decoding. For breadth-first searches, RT decoding necessitates at least an additional time delay of one block. For sequential searches such as the stack algorithm, the delay for reliable blocks may be expected to be increased compared to standard decoding (since the overhead of RTD may be significant compared to the fast sequential decoding of a reliable block), while the delay for less reliable blocks should be reduced (since the overhead plus the quick RT search should be faster than the large search required in sequential decoding of an unreliable block).

RT decoding can be considered to be a form of information-set decoding in that the symbol position reordering is used to establish a single information set. Unlike 'pure' information set

decoding, which does not allow any errors in the information set, RT decoding searches through information set errors. While other algorithms (e.g. [69][72]) are similar in that they search through error patterns using an information set determined according to bit reliabilities, RT decoding *structures* the search using a reconfigured tree or trellis. There are several advantages to this tree/trellis based search over the information set schemes. First, recall that information set schemes perform the search by altering some bits in the information set, re-encoding the codeword, computing its metric, and then repeating these steps for several patterns to be searched. The use of a tree or trellis saves effort in this search by *merging* the codewords to form a tree or trellis. This 'structuring' using the tree or trellis enables

i. branch metric calculations to be easily shared, since several codewords may descend from each contender, and

ii. error patterns to be searched based on a head metric, rather than using precomputed error patterns or using large pre-sorted lists of possible information set metrics.

A recently introduced scheme [82] is similar to RT decoding in that it reconfigures a tree in an attempt to facilitate decoding. While its tree reconfiguration avoids the computational overhead of RT decoding, the scheme has three principal disadvantages. First, it uses an unusual code tree that has a large number of branches emanating form the earliest nodes in the tree. In fact, the number of contenders for the earliest nodes in the tree will be approximately $2^{k/2}$ for randomly chosen linear block codes. This constrains the method to Gallager's 'low-density' parity check codes, which will have inferior distance properties due to the restrictions placed on the parity check matrices [82] [83]. Second, the scheme reorders the $n-k$ parity check equations; it does not reorder individual symbols as in RT decoding. It is thus constrained to consider sets of bits at a time, rather than freely selecting the order of bit processing. Third, the tree used has excess heads that do not correspond to valid codewords, so that search effort will be wasted in searching invalid sequences.

Chapter 5

Conclusion

Everything we do [in developing efficient algorithms for signal processing and coding] can be thought of in terms of the clever insertion of parentheses in a computational problem.

R.E. Blahut [84]

IN this chapter we briefly summarize and discuss the contributions reported in the previous chapters and suggest some topics for further research.

5.1 Summary of Contributions

5.1.1 Trellis Construction

Methods of trellis construction for general linear block codes were presented that are simpler than the methods of Wolf [8], Massey [9], and Forney [16]. The improved methods are based on Wolf's and Massey's methods, but are simpler in that they avoid the generation of an 'unexpurgated' trellis (which represents all uncoded sequences) followed by expurgation of non-codewords (as in [8] for general linear block codes), and they avoid matrix multiplications in forming branches of the trellis (as in [9] for non-systematic linear block codes, or as in [16] for linear block codes).

In addition to constructing a complete trellis, the methods can be used with partial trellis search algorithms to construct partial trellises 'on-the-fly', so that only the portion of the trellis that needs to be explored is generated.

5.1.2 Trellis Dimensionality

It was shown that Wolf's and Massey's trellis construction methods yield isomorphic trellises, and that these trellises are minimal. This complements Muder's result [17] that Forney's trellis construction method yields a minimal trellis. An improvement to Muder's lower bound on the maximum trellis dimension for linear block codes was found. It was also shown that the trellis dimensions remain fixed near either end of the trellis despite symbol position permutations, while in the central portion of the trellis the dimensions vary between an attainable upper bound and a lower bound. While the upper bound on the maximum trellis dimension is the familiar $\min(k, n-k)$ bound [8], the lower bound on the minimum trellis dimension in the central portion of the trellis is new. In fact this bound on the minimum trellis dimension is equal to Muder's bound on the maximum trellis dimension. The bounds indicate that only codes (and their duals) that have a smallest minimum distance $\min(d_{min}, d_{min}^{\perp})$ significantly less than the corresponding Singleton bound can possibly have a trellis with few states relative to the worst case of $\min(k, n-k)$.

5.1.3 A General Decoding Metric and its use in Trellis Decoding

The decoding problem faced by a partial trellis search was described in an alternate manner to Massey's variable-length code model [26][37]. As part of this approach, we specialized Massey's use of randomly-coded tails in order to exploit *a priori* knowledge of the information-symbol distribution and of the specific code used. It was shown that the decoder should discard heads in the obvious manner (by first exploiting merges in the trellis and then pruning the 'worst' heads first

as indicated by the metric) and that in some atypical circumstances the metric should be altered from that usually used. Such situations include unequal *a priori* information-symbol probabilities, non-linear codes, and non-symmetric, non-binary DMCs.

5.1.4 Contender Sifting

Contender sifting is the action of finding the best metrics from among a number of contenders, during a partial search of a trellis or a tree. A new breadth-first contender sifting method, *Contender Merge-Sifting* (CMS), was presented that can sift out a sorted set of the best M of $2M$ contender metrics using a worst case number of comparisons that is linear in M. In fact, for M algorithm decoding of binary linear block codes the worst case number of comparisons is precisely M. CMS offers a significant improvement over other comparison-based contender sifting methods, regardless of whether they use sorting, or selection (without regard to order within the best subset). The improvement with respect to sorting might seem surprising, since it is well known that comparison-based sorting of n items requires $O(n \log_2 n)$ comparisons. For asymptotically large M, selection schemes exist that are also linear in M, with the best of these [47] requiring a worst-case number of comparisons asymptotic to $6M$, so that CMS will be more efficient, especially for practical values of M. The advantage of CMS with respect to approximate sorting [35] is that it always delivers precisely the best set of contenders.

To attain its efficiency, CMS takes into consideration the way in which the contender metrics are formed, which *hides an inherent ordering*. During their formation the contender metrics can be easily segregated into sorted subsets, and this is exploited to achieve efficient sorting via merging. This technique is not restricted to the decoding of linear block codes. For example the contender sifting of rate 1/2 binary convolutional codes can as accomplished in at most $3M - 1$ comparisons. The important case of decoding punctured convolutional codes using CMS is also easily accommodated.

5.1.5 Reconfigurable Trellis Decoding

A soft-decision decoding method for linear block codes, referred to as *Reconfigurable Trellis Decoding*, was presented. The concept of RT decoding is to carry out a partial trellis search on a *different and more easily searched trellis.* The trellis used for decoding is dependent upon the reliability of the received data, so that it is determined 'on-the-fly'. The trellis used corresponds to the original code with its symbol positions reordered to be in most-reliable-symbol-first order. Since a partial trellis (or tree) search is used, the entire trellis or tree need not be stored or generated — only a portion of the trellis or tree is constructed as guided by the search. For trellis decoding the construction of the partial trellis is facilitated by the simplified trellis construction methods discussed in Chapter 2. For tree decoding the construction of the partial tree can use 're-encoding' of codeword heads. This is similar to 're-encoding' of codewords as used in information set decoding, except that here we do not re-encode entire codewords — rather only the codeword heads are re-encoded and extended as guided by the search.

RT decoding reduces the number of survivors retained during the search by exploiting the high reliability symbols at the beginning of the search, where a burst of poor data would otherwise require many survivors in order to retain the correct path. The symbol reordering tends to collect all of the errors into a 'burst' of unreliable symbols at the tail of the reconfigured trellis. The RT decoder can then deftly avoid retaining more survivors when processing this 'burst' since many symbols in the tail are constrained by the code. In other words, the RT decoder attempts to 'trap' the errors. In this way RT decoding avoids extensive searching through unreliable bursts of data, which are the bane of standard sequential decoding, until the constraints imposed by the reordered code can handle them.

The price paid for the more favourable search order is an additional fixed delay due to the

reordering before decoding can begin and some computational overhead to reduce the code matrix in preparation for reconfiguring the trellis or tree. This overhead and the computational effort of the search are summarized in tables of formulas that give the numbers of metric operations (comparisons, additions) and the number of binary-vector operations for RT decoding using the M algorithm. This RT-M algorithm also utilizes the Contender Merge-Sifting scheme introduced in Chapter 3. The RT-M algorithm was compared to highly-specialized ML decoders and to general linear block code decoding offered by the standard M algorithm. Compared to the most efficient ML decoders known for the binary extended Golay (24,12) code, the RT-M algorithm can attain near-ML decoding performance (estimated 0.1 dB loss) with approximately 60% of the number of metric-pair operations. Compared to standard M algorithm decoding of example codes on an AWGN channel with binary PSK and on a Rayleigh faded channel with binary NCFSK, RT decoding attained a significant reduction in computational effort.

5.1.6 Analysis of Error Probability of Pruning Decoders

The uniform error property of ML decoding of linear codes on a binary-input symmetric-output memoryless channel [52] was extended to the more general case of pruning decoding on a tree or a minimal trellis. This indicates that the error probability for pruning decoders with binary linear codes on such channels can be analyzed or simulated using only the all zero codeword.

The *pruning impact* was defined to be the increase in error probability of a pruning decoder with respect to a ML decoder. A simple upper bound on the pruning impact was found and used to estimate the pruning impact of the RT-M algorithm. This was facilitated by the introduction of an *Order Statistic Channel* (OSC) model, which enables one to represent a reordered sequence of n uses of a binary-input symmetric-output memoryless channel as a sequence of n BSCs of specific crossover probabilities. The significance of the OSC model is that it enables one to treat a reordered

block of outputs from a *soft-decision* channel as a sequence of *binary* outputs — thus avoiding the complexity of considering numerous soft-decision output values. Using the OSC model the pruning impact of the RT-M algorithm was then estimated using a very simple search procedure that is independent of the detailed structure of the code. Comparison of the estimated word error rate of the RT-M algorithm to simulation results for the binary extended Golay (24,12) code at a SNR of 2 dB (which corresponds to a word error rate of approximately 5×10^{-2}) shows agreement to within 0.25 dB, with this error diminishing for higher SNRs. The accuracy of the OSC model was argued to improve with the code length, and the accuracy of the pruning impact bound and the estimation procedure were argued to improve with increasing SNR.

5.2 Suggestions for Further Research

It would be of interest to improve the bounds on the maximum and the minimum trellis dimensions of linear block codes. In particular, it would be useful to have a good upper bound on the smallest maximum trellis dimension. This would complement the lower bound obtained here. Some other problems are perhaps best summarized by the following questions. Can one utilize more details of the weight structure of a code and its dual (i.e. besides d_{min} and $d^{\perp}_{min}$) to bound the trellis dimensions? Can one obtain tighter bounds on the trellis dimensions for specific families of codes?

Contender Merge-Sifting method could be implemented in parallel using a merging network, with a significant reduction in the network depth and number of comparators compared to a sorting network as used in [43].

What is the loss in using tree decoding instead of trellis decoding given that the same number of survivors are used? As discussed earlier, we expect this loss to be small for typical decoding situations where the number of survivors is small in relation to the number of trellis states — however a formal analysis would still be of interest. In RT decoding, the most-reliable-symbol-first reordering policy is expected to be optimal with respect to minimizing the number of branches searched for a given probability of error, if the reordering policy is constrained to be based solely on the symbol reliabilities (without regard to the effect of symbol position reordering on the trellis dimensions). Formal derivation of the optimal reordering policies in such cases are open problems.

Appendix A: Sorting Methods for Contender Sifting

A.1 Modified Insertion-Sort

The modification for insertion sorting to sift out M contenders from $2M$ is simply to retain a list of size M, instead of $2M$. The worst case number of comparisons for steps $1, 2, \ldots, M$ of the insertion-sort is

$$0 + 1 + 2 + \cdots + (M-1) = \frac{1}{2}M(M-1). \tag{A.2}$$

For the steps $M+1$ through $2M$, the worst case number of comparisons is M per step, giving $M(2M - M) = M^2$ comparisons for these steps. The total number of worst case comparisons, normalized by M, is then

$$\begin{aligned} \frac{N_c(M, 2M)}{M} &\leq \frac{\frac{1}{2}M(M-1) + M^2}{M} \\ &= \frac{1}{2}(3M - 1). \end{aligned} \tag{A.3}$$

A.2 Modified Merge-Sort

The modification for merge-sorting to sift out M contenders from $2M$ is that we need only partially merge the two sorted subsequences of length M, until the best M have been found. This final partial merging step can be done in M comparisons. To this we add the number of comparisons for two standard merge-sorts of size M.

The number of comparisons $n_c(n)$ to merge-sort an input sequence of size n (a power of 2) can be expressed as the recursion

$$n_c(n) = \begin{cases} 0 & n = 1 \\ 1 & n = 2 \\ 2n_c(n/2) + n - 1 & n \geq 2 \end{cases} \tag{A.4}$$

which can be easily verified to have the solution

$$n_c(n) = n \lg n - (n - 1) \tag{A.5}$$

where lg denotes $\log_2$.

The total number of comparisons for contender sifting is then

$$\begin{aligned} N(M, 2M) &= 2[M \lg M - (M - 1)] + M \\ &= 2M \lg M - (M - 2). \end{aligned} \tag{A.6}$$

Normalizing by M, we obtain

$$\frac{N(M, 2M)}{M} = 2 \lg M - 1 + 2/M. \tag{A.7}$$

Appendix B: Proof of Theorem 4.1

To prove the uniform error property for pruning decoders given by Theorem 4.1 we first write

$$P_{e|\mathbf{c}_j} = \int_{\mathbf{R}} Pr(\text{error} \mid \mathbf{r}, \mathbf{c}_j \text{ sent}) f(\mathbf{r} \mid \mathbf{c}_j \text{ sent}) \, d\mathbf{r} \tag{B.1}$$

and

$$P_{e|\mathbf{c}_k} = \int_{\mathbf{R}} Pr(\text{error} \mid \mathbf{r}', \mathbf{c}_k \text{ sent}) f(\mathbf{r}' \mid \mathbf{c}_k \text{ sent}) \, d\mathbf{r}'. \tag{B.2}$$

To show that (B.1) and (B.2) are equal, it is sufficient to show that for each $\mathbf{r}$ there is a corresponding $\mathbf{r}'$ such that

$$f(\mathbf{r}' \mid \mathbf{c}_k \text{ sent}) = f(\mathbf{r} \mid \mathbf{c}_j \text{ sent}) \tag{B.3}$$

and

$$Pr(\text{error} \mid \mathbf{r}', \mathbf{c}_k \text{ sent}) = Pr(\text{error} \mid \mathbf{r}, \mathbf{c}_j \text{ sent}). \tag{B.4}$$

Now $f(\mathbf{r} \mid \mathbf{c}_j \text{ sent}) = \prod_{i=1}^{n} f(r_i \mid c_{ji})$ and $f(\mathbf{r} \mid \mathbf{c}_k \text{ sent}) = \prod_{i=1}^{n} f(r_i \mid c_{ki})$. By choosing

$$r_i' = \begin{cases} r_i & , c_{ki} = c_{ji} \\ -r_i & , c_{ki} \neq c_{ji} \end{cases} \tag{B.5}$$

it is easy to see that (B.3) will be satisfied, since for each position where $\mathbf{c}_j$ and $\mathbf{c}_k$ agree we have

$$f(r_i' \mid c_{ki}) = f(r_i \mid c_{ki}) = f(r_i \mid c_{ji}) \tag{B.6}$$

and for each position where $\mathbf{c}_j$ and $\mathbf{c}_k$ disagree we have, using the symmetry of the binary-input symmetric-output channel (i.e. $f(r_i \mid 0) = f(-r_i \mid 1)$), that

$$f(r_i' \mid c_{ki}) = f(-r_i \mid \overline{c_{ji}}) = f(r_i \mid c_{ji}). \tag{B.7}$$

To show that (B.4) holds, we first let $\widehat{\mathbf{c}}_j$ denote the output codeword when $\mathbf{c}_j$ was transmitted and $\mathbf{r}$ was received, and similarly we let $\widehat{\mathbf{c}}_k$ denote the output codeword when $\mathbf{c}_k$ was transmitted and $\mathbf{r}'$ was received. Also, we let $\mathbf{e} = \mathbf{c}_j + \widehat{\mathbf{c}}_j$ denote the error vector between the transmitted codeword $\mathbf{c}_j$ and the output codeword $\widehat{\mathbf{c}}_j$, and similarly we let $\mathbf{e}' = \mathbf{c}_k + \widehat{\mathbf{c}}_k$ denote the error vector between the transmitted codeword $\mathbf{c}_k$ and the output codeword $\widehat{\mathbf{c}}_k$. To show that (B.4) holds, it is sufficient to show that $\mathbf{e} = \mathbf{e}'$.

If $\mathbf{e}$ is to equal $\mathbf{e}'$, we need $\mathbf{c}_j + \widehat{\mathbf{c}}_j = \mathbf{c}_k + \widehat{\mathbf{c}}_k$, or equivalently we need $\mathbf{c}_j + \mathbf{c}_k = \widehat{\mathbf{c}}_j + \widehat{\mathbf{c}}_k$. Define $\mathbf{z} = \mathbf{c}_j + \mathbf{c}_k$. We then need only to show that the output codewords also differ by $\mathbf{z}$. For this it is sufficient to show that at any decoding stage that each head $\mathbf{c}^h$ retained by the decoder using $\mathbf{r}$ has a corresponding head $\mathbf{c}'^h$ retained by the decoder using $\mathbf{r}'$, where $\mathbf{c}^h + \mathbf{c}'^h = \mathbf{z}^h$. We will show this by showing that each head that enters any node σ in the trellis when $\mathbf{r}$ is processed has a counterpart that enters a node σ' when $\mathbf{r}'$ is processed, where the corresponding heads differ by $\mathbf{z}^h$, and where the corresponding heads have identical metrics. In other words, we aim to show that the decoder encounters the same set of metrics at a node σ when processing $\mathbf{r}$ as it does at σ' when processing $\mathbf{r}'$, with the heads for these 'equal-metric counterparts' differing by $\mathbf{z}^h$.

Recall that the decoding rule is to minimize (3.19)

$$\sum_{\mathbf{c}^h \in S^h_{P_m}} \frac{f(\mathbf{r}^h \mid \mathbf{c}^h) Pr(\mathbf{c}^h)}{f_0(\mathbf{r}^h)}. \tag{B.8}$$

Since the heads entering any given node have equal lengths, and the decoder is taken to assume equiprobable codewords, for the purpose of comparing metrics at a depth we can simply use $f(\mathbf{r}^h \mid \mathbf{c}^h)$.

Consider a head metric $f(\mathbf{r}^h \mid \mathbf{c}^h)$. For a head metric using $\mathbf{r}'$ to equal this we need

$$f\left(\mathbf{r}'^h \mid \mathbf{c}'^h\right) = f\left(\mathbf{r}^h \mid \mathbf{c}^h\right). \tag{B.9}$$

Recall (from (B.5) and the definition of $\mathbf{z}$) that we have chosen $\mathbf{r}'$ to be the negative of $\mathbf{r}$ in those positions where $\mathbf{z}$ is nonzero. Using this fact and the symmetry of the binary-input symmetric-output channel, for (B.9) to hold we will need $\mathbf{c}'^h$ to differ from $\mathbf{c}^h$ in those positions where $\mathbf{z}^h$ is nonzero. In other words we must have that $\mathbf{c}^h + \mathbf{c}'^h = \mathbf{z}^h$, which is part of what we needed to show. All that remains is to show that *all* the heads that merge at a node σ (when processing $\mathbf{r}$) have their equal-metric counterparts (when processing $\mathbf{r}'$) at a node σ'. From Chapter 2 (see Proposition 2.1 on pg. 26 and (2.7)), each head that merges at a node σ in a minimal trellis satisfies

$$\sigma = \mathbf{H}^h \mathbf{c}^{h\mathrm{T}}. \tag{B.10}$$

Now, since any head $\mathbf{c}'^h$ that is an equal-metric counterpart to $\mathbf{c}^h$ satisfies $\mathbf{c}'^h = \mathbf{c}^h + \mathbf{z}^h$, we have that its state is

$$\begin{aligned} \sigma' &= \mathbf{H}^h \ \mathbf{c}'^{hT} \\ &= \mathbf{H}^h \left(\mathbf{c}^{hT} + \mathbf{z}^{hT} \right) \\ &= \sigma + \mathbf{H}^h \mathbf{z}^{hT}. \end{aligned} \tag{B.11}$$

Now either $\mathbf{H}^h \mathbf{z}^{hT}$ is nonzero or it is zero. In the former case, we have that all the heads that merge at σ when $\mathbf{r}$ is processed have their equal-metric counterparts merge at some other node σ', as required. In the latter case, we have a similar situation except that $\sigma' = \sigma$.

Using any minimal-trellis state-assignment method besides that used above [8] will yield an isomorphic trellis [17]. However, this relabeling of trellis states will not alter our result, since the state labels are immaterial to the head metrics and the partitioning of the heads into states. Finally, we comment that above proof holds for symmetrically quantized outputs, under the assumption that metric ties are randomly resolved, as is assumed in the proof of the uniform error property for ML decoding in [52].

References

[1] A. J. Viterbi, "Wireless digital communications: A view based on three lessons learned," *IEEE Commun. Magazine*, vol. 29, pp. 33–36, Sept. 1991.

[2] R. G. Gallager, *Information Theory and Reliable Communication*. Wiley, 1968.

[3] J. M. Wozencraft and I. M. Jacobs, *Principles of Communication Engineering*. Wiley, 1965.

[4] J. L. Massey, "The how and why of channel coding," in *IEEE Int. Zurich. Seminar on Digital Communications*, 1984.

[5] G. D. Forney, *Concatenated Codes*. M.I.T. Press, 1966.

[6] D. Chase, "A class of algorithms for decoding block codes with channel measurement information," *IEEE Trans. Inform. Theory*, vol. IT-18, pp. 170–182, Jan. 1972.

[7] W. Jakes, ed., *Microwave Mobile Communications*. Wiley, 1974.

[8] J. K. Wolf, "Efficient maximum likelihood decoding of linear block codes using a trellis," *IEEE Trans. Inform. Theory*, vol. IT-24, pp. 76–80, Jan. 1978.

[9] J. L. Massey, "Foundations and methods of channel coding," in *Proc. of the Int. Conf. on Info. Theory and Systems*, vol. 65, NTG-Fachberichte, Sept. 1978.

[10] G. D. Forney, "Review of random tree codes," tech. rep., NASA Ames Research Center, Moffet Field CA, Dec. 1967. Contract NAS2–3637 NASA CR73176 Final Report, App.A.

[11] G. D. Forney, "The Viterbi algorithm," *Proc. IEEE*, vol. 61, pp. 268–278, 1973.

[12] A. J. Viterbi, "Error bounds for convolutional code and an asymptotically optimal decoding algorithm," *IEEE Trans. Inform. Theory*, vol. IT-13, pp. 260–269, Apr. 1967.

[13] J. K. Omura, "On the Viterbi algorithm," *IEEE Trans. Inform. Theory*, vol. IT-15, pp. 177–179, Jan. 1969.

[14] G. Ungerboeck, "Channel coding with multilevel/phase signals," *IEEE Trans. Inform. Theory*, vol. IT-28, no. 1, pp. 55–67, 1982.

[15] L. R. Bahl, J. Cocke, F. Jelinek, and J. Raviv, "Optimal decoding of linear codes for minimizing symbol error rate," *IEEE Trans. Inform. Theory*, pp. 284–287, Mar. 1974.

[16] G. D. Forney, "Coset codes — Part II: Binary lattices and related codes," *IEEE Trans. Inform. Theory*, vol. IT-34, pp. 1152–1187, Sept. 1988. Appendix A.

[17] D. J. Muder, "Minimal trellises for block codes," *IEEE Trans. Inform. Theory*, vol. IT-34, pp. 1049–1053, Sept. 1988.

[18] J. B. Cain, G. C. Clark, and J. M. Geist, "Punctured convolutional codes of rate (n-1)/n and simplified maximum likelihood decoding," *IEEE Trans. Inform. Theory*, vol. IT-25, pp. 97–100, Apr. 1979.

[19] J. Hagenauer, "Rate compatible punctured convolutional codes (RCPC-codes) and their application," *IEEE Trans. Commun.*, vol. 36, pp. 389–400, Apr. 1988.

[20] D. M. Mandelbaum, "An adaptive-feedback coding scheme employing incremental redundancy," *IEEE Trans. Inform. Theory*, vol. IT-20, pp. 388–389, May 1974.

[21] G. Strang, *Linear Algebra and its Applications*. Academic Press, 1976.

[22] K. R. Matis and J. W. Modestino, "Reduced-search soft-decision trellis decoding of linear block codes," *IEEE Trans. Inform. Theory*, vol. IT-28, pp. 349–355, Mar. 1982.

[23] T. Kasami, T. Takata, T. Fujiwara, and S. Lin, "On complexity of trellis structure of linear block codes," in *IEEE Int. Symposium on Information Theory*, June 1991.

[24] R. Bellman, *Adaptive Control Processes: A Guided Tour*. Princeton University Press, 1961.

[25] R. Bellman, *Applied Dynamic Programming*. Princeton University Press, 1962.

[26] J. L. Massey, "Variable-length codes and the Fano metric," *IEEE Trans. Inform. Theory*, vol. IT-18, pp. 196–198, Jan. 1972.

[27] D. E. Knuth, *The Art of Computer Programming: Volume III, Sorting and Searching*. Addison-Wesley, 1973.

[28] R. Blahut, *Digital Transmission of Information*. Addison-Wesley, 1990.

[29] J. B. Anderson and S. Mohan, "Sequential decoding algorithms: A survey and cost analysis," *IEEE Trans. Commun.*, vol. COM-32, pp. 169–176, Feb. 1984.

[30] F. Jelinek and J. B. Anderson, "Instrumentable tree encoding of information sources," *IEEE Trans. Inform. Theory*, pp. 118–119, Jan. 1971.

[31] J. B. Anderson and J. B. Bodie, "Tree encoding of speech," *IEEE Trans. Inform. Theory*, vol. IT-21, pp. 379–387, July 1975.

[32] S. J. Simmons, "Breadth-first trellis decoding with adaptive effort," *IEEE Trans. Commun.*, vol. COM-38, pp. 3–12, Jan. 1990.

[33] R. M. Fano, "A heuristic discussion of probabilistic decoding," *IEEE Trans. Inform. Theory*, vol. IT-19, pp. 64–74, Apr. 1963.

[34] K. Zigangirov, "Some sequential decoding procedures," *Probl. Peredach. Inform.*, vol. 2, pp. 13–25, 1966.

[35] F. Jelinek, "A fast sequential decoding algorithm using a stack," *IBM J. Res. Dev.*, vol. 13, pp. 675–685, Nov. 1969.

[36] I. M. Jacobs and E. R. Berlekamp, "A lower bound to the distribution of computation for sequential decoding," *IEEE Trans. Inform. Theory*, vol. IT-13, pp. 167–174, Apr. 1967.

[37] J. L. Massey, "Error bounds for tree codes, trellis codes, and convolutional codes with encoding and decoding procedures," In *Coding and Complexity*, G. Longo (Ed.) Springer-Verlag, 1976.

[38] J. B. Anderson, "Limited search trellis decoding of convolutional codes," *IEEE Trans. Inform. Theory*, vol. IT-35, pp. 944–955, Sept. 1989.

[39] T. Hashimoto, "A list-type reduced constraint generalizaton of the Viterbi algorithm," *IEEE Trans. Inform. Theory*, vol. IT–33, pp. 866–876, Nov. 1987.

[40] D. Haccoun and M. J. Ferguson, "Generalized stack algorithms for decoding convolutional codes," *IEEE Trans. Inform. Theory*, vol. IT–21, pp. 638–651, Nov. 1975.

[41] A. Aho, J. Hopcroft, and J. Ullman, *Data Structures and Algorithms*. Addison-Wesley, 1982.

[42] S. Mohan and A. K. Sood, "A multiprocessor architecture for the (M,L)-algorithm suitable for VLSI implementation," *IEEE Trans. Commun.*, vol. COM-34, pp. 1218–1224, Dec. 1986.

[43] S. J. Simmons, "A bitonic-sorter based VLSI implementation of the M algorithm," in *IEEE Pacific Rim Conference on Communications, Computers and Signal Processing*, (Victoria, B.C., Canada), pp. 337–340, June 1989.

[44] S. J. Simmons, "A nonsorting VLSI structure for implementing the (M,L) algorithm," *IEEE J. Selected Areas Commun.*, vol. JSAC-6, pp. 538–546, Apr. 1988.

[45] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*. McGraw-Hill, 1990.

[46] M. Blum, R. Floyd, V. Pratt, R. Rivest, and R. Tarjan, "Time bounds for selection," *J. Comp. Sys. Sci.*, vol. 7, pp. 448–461, 1973.

[47] A. Schönhage, M. Paterson, and N. Pippenger, "Finding the median," *J. Comp. Sys. Sci.*, vol. 13, pp. 184–199, 1976.

[48] R. E. Blahut, *Theory and Practice of Error Control Codes*. Addison-Wesley, 1983.

[49] F. Hemmati, "Bidirectional decoding of linear block codes," in *IEEE Sympos. Inf. Theory*, 1990.

[50] D. Haccoun and G. Belzile, "Bidirectional algorithms for the decoding of convolutional codes," in *IEEE Inform. Theory Symposium*, p. 177, 1990.

[51] K. Li and S. Kallel, "Bidirectional sequential decoding for convolutional codes," in *IEEE Pacific Rim Conf. on Communications, Computers, and Signal Processing*, pp. 200–204, 1991.

[52] A. J. Viterbi and J. Omura, *Principles of Digital Communication and Coding*. McGraw-Hill, 1979.

[53] R. Silverman and M. Balser, "Coding for constant-data-rate systems," *IRE Trans. Inform. Theory*, vol. PG IT-4, pp. 50–63, 1954.

[54] G. Einarsson and C. Sundberg, "A note on soft decision decoding with successive erasures," *IEEE Trans. Inform. Theory*, vol. IT-22, pp. 88–96, Jan. 1976.

[55] C. Yu and D. Costello, "Generalized minimum distance decoding algorithms for Q-ary output channels," *IEEE Trans. Inform. Theory*, vol. IT-26, pp. 238–243, Mar. 1980.

[56] D. Taipale and M. Pursley, "An improvement to generalized-minimum-distance decoding," *IEEE Trans. Inform. Theory*, vol. IT-37, pp. 167–172, Jan. 1991.

[57] E. Berlekamp, "The technology of error-correcting codes," *Proc. IEEE*, vol. 68, pp. 564–593, May 1980.

[58] E. Weldon, "Decoding binary block codes on Q-ary output channels," *IEEE Trans. Inform. Theory*, vol. IT-17, pp. 713–718, Nov. 1971.

[59] S. Wainberg and J. Wolf, "Algebraic decoding of block codes over a q-ary input, Q-ary output channel, Q>q," *Information and Control*, vol. 22, pp. 232–247, Nov. 1973.

[60] C. Hackett, "An efficient algorithm for soft-decision decoding of the (24,12) extended Golay code," *IEEE Trans. Commun.*, vol. COM-29, pp. 909–911, June 1981.

[61] J. Massey, *Threshold Decoding*. MIT Press, 1963.

[62] J. Conway and N. Sloane, "Soft decoding techniques for codes and lattices, including the Golay code and the Leech lattice," *IEEE Trans. Inform. Theory*, vol. IT–32, pp. 41–56, Jan. 1986.

[63] Y. Be'ery and J. Snyders, "Optimal soft decision block decoders based on fast Hadamard transform," *IEEE Inf. Theory*, vol. IT-32, pp. 355–364, May 1986.

[64] J. Snyders and Y. Be'ery, "Maximum likelihood soft decoding of binary block codes and decoders for Golay codes," *IEEE Trans. Inform. Theory*, vol. IT–35, pp. 963–975, Sept. 1989.

[65] J. Snyders, "Reduced lists of error patterns for maximum likelihood soft decoding," *IEEE Trans. Inform. Theory*, vol. IT–37, pp. 1194–1200, July 1991.

[66] E. Prange, "The use of information sets in decoding cyclic codes," *IRE Trans. Inform. Theory*, vol. IT-8, pp. s5–s9, 1962.

[67] F. MacWilliams, "Permutation decoding of systematic codes," *Bell Syst. Tech. J.*, vol. 43, pp. 485–505, 1964.

[68] J. Schönheim, "On coverings," *Pac. J. Math*, vol. 14, pp. 1405–1411, 1964.

[69] G. Clark and J. B. Cain, *Error-Correction Coding for Digital Communications*. Plenum, 1981.

[70] J. Coffey and R. M. Goodman, "The complexity of information set decoding," *IEEE Trans. Inform. Theory*, vol. IT–36, pp. 1031–1037, Sept. 1990.

[71] T. Kasami, "A decoding procedure for multiple-error-correcting cyclic codes," *IEEE Trans. Inform. Theory*, vol. IT-10, pp. 134–138, 1964.

[72] B. Dorsch, "A decoding algorithm for binary block codes and J–ary output channels," *IEEE Trans. Inform. Theory*, vol. IT–20, pp. 391–394, May 1974.

[73] C. Lin and J. Anderson, "M-algorithm decoding of channel convolutional codes," in *Conf. on Information Science and Systems*, (Princeton University), pp. 362–366, 1986.

[74] G. Pottie and D. Taylor, "A comparison of reduced complexity decoding algorithms for trellis codes," *IEEE J. Selected Areas Commun.*, vol. COM-38, pp. 981–991, July 1990.

[75] T. Matsumoto, "Trellis decoding of linear block codes in digital mobile radio," in *IEEE Vehic. Tech. Conf.*, pp. 6–11, 1988.

[76] H. A. David, *Order Statistics*. Wiley, 1981.

[77] J. Sanie, K. D. Donohue, and N. M. Bilgutay, "Order statistic filters as postdetection processors," *IEEE Trans. Acoust. Speech Sig. Proc.*, vol. 38, pp. 1722–1731, Oct. 1990.

[78] A. Vardy and Y. Be'ery, "More efficient decoding of the Golay codes," *IEEE Trans. Inform. Theory*, vol. IT–37, pp. 667–672, May 1991.

[79] J. Proakis, *Digital Communication*. McGraw-Hill, 2 ed., 1983.

[80] R. Clarke, "A statistical theory of mobile radio reception," *Bell Syst. Tech. J.*, vol. 47, pp. 957–1000, 1968.

[81] A. Kot and C. Leung, "Optimal partial decision combining in diversity systems," *IEEE Trans. Commun.*, vol. COM-38, pp. 981–991, July 1990.

[82] B. Radosavljevic, E. Arikan, and B. Hajek, "Sequential decoding of low-density parity-check codes by adaptive reordering of parity checks," *IEEE Trans. Inform. Theory*, vol. IT-38, pp. 1833–1839, Nov. 1992.

[83] R. Gallager, *Low-Density Parity-Check Codes*. MIT Press, 1963.

[84] R. Blahut, *Fast Algorithms for Digital Signal Processing*. Addison-Wesley, 1985.