A GENERALIZED POST-DETECTOR COMPATIBLE SOFT-OUTPUT VITERBI ALGORITHM (SOVA)

By

David Kwan

B.A.Sc, The University of British Columbia, 1989

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF

THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF APPLIED SCIENCE (M.A.SC)

in

THE FACULTY OF GRADUATE STUDIES

ELECTRICAL ENGINEERING

We accept this thesis as conforming

to the required standard

THE UNIVERSITY OF BRITISH COLUMBIA

April 1996

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.

Department of Elec. Eng. (Ap.Sc.)

The University of British Columbia
Vancouver, Canada

Date April 25, 1996

Abstract

A generalized soft-output Viterbi algorithm (SOVA) that is applicable to any (n, k, m) convolutional code is proposed. The algorithm is compatible with the post-detector architecture proposed by Berrou *et al.* thereby achieving low computational complexity. By starting with Battail's generalized revision algorithm and re-referencing the relative values to the surviving path to each state, significant simplifications are made possible. By comparing the resultant simplified revision equation for $(n, 1, m)$ convolutional codes with Berrou's proposed post-detector compatible algorithm it is possible to deduce the additional modifications necessary to arrive at a (n, k, m) post detector compatible algorithm. Simulations show that with a revision depth greater than five times a code's constraint length, the proposed algorithm is capable of producing relatively high quality a posteriori input symbol estimates.

Table of Contents

List of Tables

List of Figures

Dedication

In memory of my grandmother and father, and for my ever persevering mother.

Acknowledgements

The author would like to thank his advisor Dr. Samir Kallel for proposing such an interesting project. The many insightful discussions, his guidance, and encouragement have been invaluable. The author would also like to thank Ian Marsland for proof reading the thesis and for his help throughout the project. Finally, credit has to be given to Abderrazek Zaafrani for his invaluable English translation of Battail's paper [2].

Chapter 1

Introduction

1.1 Motivation and Scope of the Thesis

There arises many situations in communications where the serial concatenation of several codes may transpire. For such systems, optimum performance is achieved when the decoding is performed on the resulting effective global code. However, if the global code is too complex, decoding in this manner may not be viable. An alternative is to decode each individual code separately using the results from each decoding stage as input for the subsequent decoding stage. For this arrangement to achieve its full performance potential it is necessary that all of the decoding stages perform soft-decision maximum a posteriori (MAP) decoding. Unfortunately, many of the popular decoding algorithms such as the Viterbi algorithm output hard-decisions. Therefore, if any of these algorithms are used for decoding one of the inner codes, this will constrain the subsequent decoding stage to hard-decision decoding resulting in a degradation of performance. As shall be shown later, it is possible for a subsequent decoding stage to perform soft-decision decoding if the preceding decoding stage can provide a posteriori input symbol probabilities instead of hard decisions. Decoders that are capable of providing such information are usually referred to as "soft-output" decoders. Applications that can benefit from the use of "soft-output" decoders include the serial concatenation of codes to produce a more powerful, yet still easily decoded, global code. Other examples would be the use of Viterbi equalization for previously encoded data, or the transmission of

previously encoded data using TCM. A more recent example would be the iterative "Turbo" decoding algorithm for use with parallel concatenated codes [4]. Motivated by the potential gains that soft-output decoding can bring to these applications (for Turbo decoding, it is in-fact an essential element) much research has been done in the area of finding fast, efficient, and accurate soft-output decoding algorithms. The work described in this thesis is part of that effort.

One of the algorithms that can be employed for soft-output decoding is the symbol-by-symbol MAP algorithm. Proposed in [1] as an optimal method of decoding convolutional codes, its use for soft-output decoding was probably never intended. However, it does produce as a natural by-product of the decoding process the necessary a posteriori input symbol probabilities required for soft-output decoding. Its main advantage is that it deduces the exact value for the a posteriori input probabilities. Its disadvantage is that it requires a relatively large amount of computation. Indeed, this is one of the reasons it is not commonly used for decoding convolutional codes – most designers opt in favour of the Viterbi algorithm. In addition to the relatively high computational requirements, the decoding delay and the amount of storage necessary to implement the algorithm are dependent upon the length of the transmitted code sequence. Several sub-optimal variants of this algorithm have been proposed to address some of these problems. For example, to reduce the amount of computation necessary there is the "Best-Path" algorithm described in [7]. To eliminate the dependence of the decoding delay and required storage on the length of the transmitted code sequence, there is the algorithm proposed in [14].

An alternative approach that may be used for soft output decoding is based upon a heuristic modification to the standard Viterbi algorithm. The resulting soft-output Viterbi algorithms (SOVA) [2][3][6] are able to deduce estimates of the a posteriori input

probabilities.[1] One of the main attractions of this approach is that because the resulting algorithms are based on the Viterbi algorithm, they have the desirable quality that the decoding delay and required storage are independent of the length of the transmitted code sequence. They are also computationally efficient in that the majority of the operations needed to implement these algorithms involve relatively simple operations such as comparisons and additions. Finally, as shown in [3], if one can derive a SOVA that is compatible with the "Post-detector" architecture, then it is possible to significantly reduce the amount of required computation and storage needed to estimate the a posteriori input probabilities. In the "Post-detection" scheme, the soft-output decoder is given knowledge a priori of what its eventual surviving path shall be. If the algorithm satisfies certain conditions (described in Chapter-2), it may use this information to its advantage thereby eliminating many of the required operations.

The work described in this thesis focuses on the heuristic approach to deducing a posteriori input probabilities. At the onset of this project, one major short coming to the SOVA approach to generating soft output information was that there did not seem to exist an efficient algorithm that was applicable to any (n, k, m)[2] code. All of the algorithms proposed in [2][3][6] are only applicable to binary input $(n, 1, m)$ type codes. If it was desired to use any of these algorithms for a rate-k/n code then such a code would have to be synthesized by puncturing a rate-$1/n$ code. Therefore, the main objective of the research documented by this thesis was to find a heuristic based soft-output decoding algorithm that is applicable to any (n, k, m) type code. For the sake of computational efficiency it was also desirable that this algorithm be compatible with the post-detector

[1]The term SOVA is often used in reference to the specific algorithm proposed by Hagenauer and Hoeher in [6]. However, due to the similar foundations, for this thesis the term SOVA is used in reference to any algorithm that is based upon a heuristic modification of the standard Viterbi algorithm.

[2]Following the notation as presented in [11], a (n, k, m) code represents a convolutional code with n-output bits, k-input bits, and a memory of m-bits.

architecture proposed in [3]. Having found such an algorithm, its performance was evaluated through the use of computer simulations of various concatenated systems. Its computational complexity was also compared with the MAP and Best-Path algorithms.

1.2 Outline of the Thesis

This thesis is organized as follows. In chapter-2, the use of a posteriori input probabilities for soft decision decoding is described. This is followed by a brief description of the algorithms proposed in [2][3][6] – on which the derivation described in Chapter-3 depends. In Chapter-3 the newly proposed "generalized post-detector algorithm" is derived. The computational complexity is compared with MAP and Best-Path algorithms. A brief discussion of some of the potential ramifications of the approximations made during the algorithm's derivation is presented. Computer simulation results are shown in Chapter-4. In Chapter-5 a summary is presented and suggestions for possible further investigation are made.

Chapter 2

Background and Underlying Principles

2.1 Using Soft-Outputs in Serially Concatenated Systems

Consider the serially concatenated system shown in Figure 2.1. In this system a message sequence $\bar{m}$ is multiplexed into several sub-messages $\bar{m}_i$. These sub-messages are encoded using the outer code. The resulting code sequences $\bar{p}_i$ are then interleaved resulting in the sequences $\bar{q}_j$. The sequences $\bar{q}_j$ are encoded using the inner code and transmitted over an AWGN channel. The received sequences $\bar{R}_j$ are decoded resulting in the sequences $\bar{U}_j$. These sequences are then de-interleaved into the sequences $\bar{Y}_i$. The purpose of the interleaver/de-interleaver operation is two-fold. First, it eliminates any correlation in the symbols of each sequences $\bar{q}_j$. Second, it eliminates any correlation in the "noise" samples in each received sequences $\bar{Y}_i$. Finally, the sequences $\bar{Y}_i$ are then decoded into the sub-messages $\bar{m}'_i$.

For such a system to achieve optimum performance, both the inner and outer decoders should perform soft-decision MAP decoding. Clearly soft-decision decoding for the inner decoder does not pose any problems since the received sequences $\bar{R}_j$ represent sampled continuous random processes. In addition, since the symbols from the inner decoder are transmitted over an AWGN channel, the conditional density function of the received symbols is well known:

$$f(R_{ji}|S_{ji}) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{|R_{ji} - s_{ji}|^2}{2\sigma^2}\right) \tag{1}$$

where $\bar{s}_j$ is the transmitted signal sequence and σ^2 is the noise variance of the AWGN

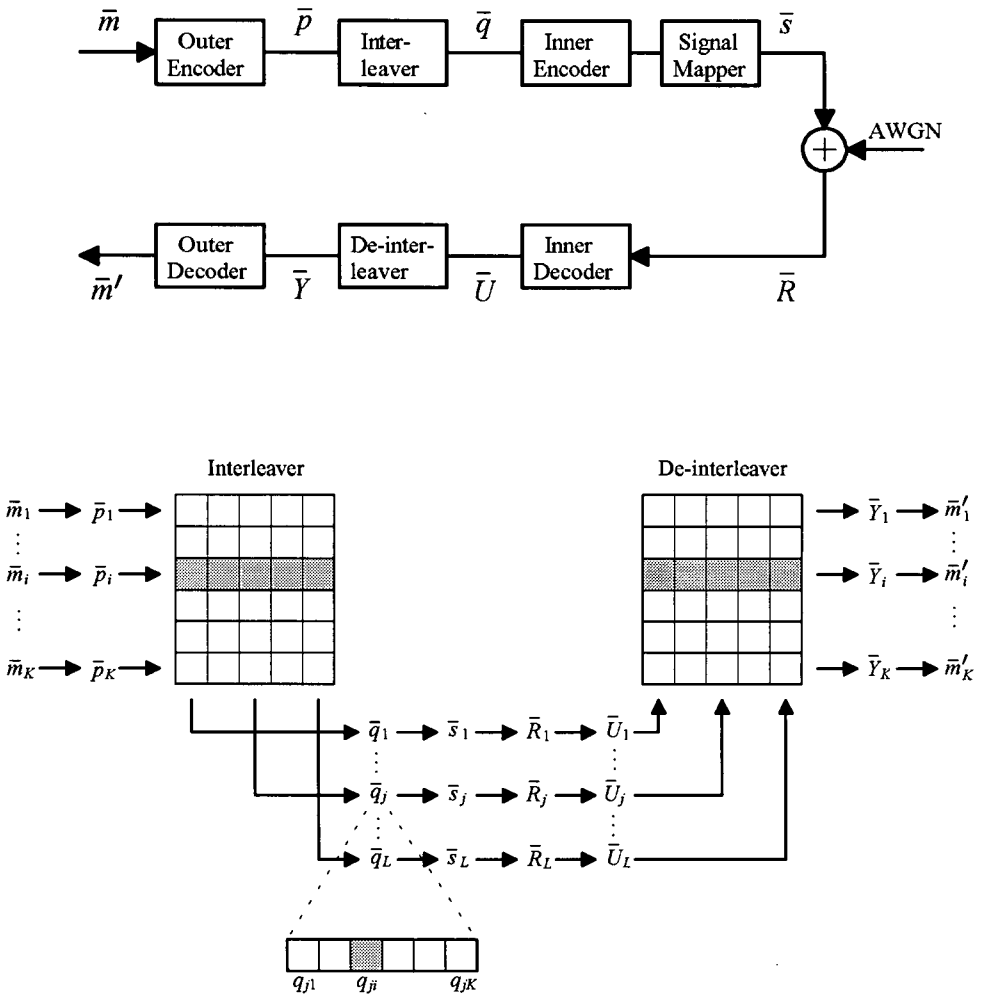


Figure 2.1: Concatenated System with Corresponding Data Flow Diagram.

channel. This conditional density function may be used to perform MAP decoding utilizing the unquantized input symbols. Soft-decision decoding for the outer code, however, is not as straight forward since any conventional decoder structure used for the inner decoder will deliver hard quantized decisions.

To determine how one can perform soft-decision MAP decoding for the outer code assume, for the time being, the existence of a soft-output inner decoder that will output an unquantized sequence $\bar{U}_j$ whose samples U_{ji} (and hence Y_{ij}) are suitable for any outer decoder to perform soft-decision decoding. To satisfy the condition that the outer decoder perform MAP sequence decoding it must utilize the following selection criteria:

$$\text{Find } \bar{m}_i' \text{ such that } \quad P(\bar{m}_i'|\bar{Y}_i) > P(\bar{m}_i|\bar{Y}_i) \quad \forall \quad \bar{m}_i \neq \bar{m}_i' \tag{2}$$

Substituting

$$P(\bar{m}|\bar{Y}) = \frac{f(\bar{Y}|\bar{m})P(\bar{m})}{f(\bar{Y})} \tag{3}$$

and asserting the condition that all sequences $\bar{m}$ are equally likely, then (2) may be rewritten as:

$$\text{Find } \bar{m}_i' \text{ such that } \quad f(\bar{Y}_i|\bar{m}_i') > f(\bar{Y}_i|\bar{m}_i) \quad \forall \quad \bar{m}_i \neq \bar{m}_i' \tag{4}$$

Because of the one-to-one relationship between the message sequences $\bar{m}$ and the codewords $\bar{p}$ this is equivalent to:

$$\text{Find } \bar{m}_i' \text{ such that } \quad f(\bar{Y}_i|\bar{p}_i(\bar{m}_i')) > f(\bar{Y}_i|\bar{p}_i(\bar{m}_i)) \quad \forall \quad \bar{m}_i \neq \bar{m}_i' \tag{5}$$

where $\bar{p}_i(\bar{m}_i)$ represents the specific code sequence $\bar{p}_i$ associated with the message sequence $\bar{m}_i$. Defining the "error" sequence $\bar{\eta}_i = \bar{Y}_i - \bar{p}_i(\bar{m}_i)$, each side of (5) may be expressed as,

$$\begin{aligned} f(\bar{Y}_i|\bar{p}_i(\bar{m}_i)) &= f(\bar{\eta}_i = \bar{Y}_i - \bar{p}_i(\bar{m}_i)) \\ &= \prod_{j=1}^{L} f(\eta_{ij} = Y_{ij} - p_{ij}(\bar{m}_i)) \qquad \text{due to interleaving} \\ &= \prod_{j=1}^{L} f(Y_{ij}|p_{ij}(\bar{m}_i)) \end{aligned} \tag{6}$$

Because $\bar{q}$ and $\bar{U}$ are merely $\bar{p}$ and $\bar{Y}$ interleaved respectively,

$$\begin{aligned} f(\bar{Y}_i|\bar{p}_i(\bar{m}_i)) &= \prod_{j=1}^{L} f(U_{ji}|q_{ji}(\bar{m}_i)) \\ &= \prod_{j=1}^{L} \frac{P(q_{ji}(\bar{m}_i)|U_{ji})f(U_{ji})}{P(q_{ji}(\bar{m}_i))} \end{aligned} \tag{7}$$

Therefore (5) maybe rewritten as:

Find $\bar{m}_i'$ such that,

$$\prod_{j=1}^{L} \frac{P(q_{ji}(\bar{m}_i')|U_{ji})P(U_{ji})}{P(q_{ji}(\bar{m}_i'))} > \prod_{j=1}^{L} \frac{P(q_{ji}(\bar{m}_i)|U_{ji})P(U_{ji})}{P(q_i(\bar{m}_i))} \quad \forall \quad \bar{m}_i \neq \bar{m}_i' \tag{8}$$

$P(U_{ji})$ does not depend on $\bar{m}_i$ and therefore can be cancelled from both sides. If the inner code has the property that $P(q_{ji}(\bar{m}_i))$ is equal for all choices of q_{ji} then it too can be eliminated. Thus the only term that the outer decoder needs to perform MAP soft-decision decoding are the probabilities $P(q_{ji}|U_{ji})$.

Because of interleaving each of the q_{ji} of the sequence $\bar{q}_j$ are independent. Therefore knowledge of U_{jx} ($\forall x \neq i$) will have no affect on $P(q_{ji}|U_{ji})$. Consequently, it is possible to substitute $P(q_{ji}|U_{ji})$ for $P(q_{ji}|\bar{U}_j)$. The resulting MAP decision criteria becomes:

$$\text{Find } \bar{m}_i' \text{ such that } \prod_{j=1}^{L} P(q_{ji}(\bar{m}_i')|\bar{U}_j) > \prod_{j=1}^{L} P(q_{ji}(\bar{m}_i)|\bar{U}_j) \quad \forall \quad \bar{m}_i \neq \bar{m}_i' \tag{9}$$

The necessity of this substitution shall be shown in the following paragraph.

Previously, the assumption of the existence of a soft-output inner decoder that delivers unquantized decoded information sequences $\bar{U}$ was made. Assume that each of these decoded sequences corresponds to a unique received sample sequence $\bar{R}$ resulting in a one-to-one relationship between $\bar{U}$ and $\bar{R}$. Under these circumstances, $P(q_{ji}|\bar{U}_j)$ will be equal

to $P(q_{ji}|\bar{R}_j)$ since knowing $\bar{U}_j$ is equivalent to knowing $\bar{R}_j$ (and vice versa). Note however that the procedure for calculating $P(q_{ji}|\bar{R}_j)$ is already well known since these are the exact quantities that must be computed if the inner decoder were a symbol-by-symbol MAP decoder. Therefore, if the outer decoder were to set $P(q_{ji}|\bar{U}_j)$ to $P(q_{ji}|\bar{R}_j)$ as calculated by an inner MAP decoder, this would be equivalent to using the aforementioned fictitious soft-output inner decoder. Since the outputs from this fictitious decoder are unquantized and there is no information reduction going from $\bar{R}$ to $\bar{U}$, the outer decoder will be performing soft-decision decoding.

As a result of the discussions presented in the previous paragraphs it can be concluded that the decision rule:

$$\text{Find } \bar{m}_i' \text{ such that } \prod_{j=1}^{L} P(q_{ji}(\bar{m}_i')|\bar{R}_j) > \prod_{j=1}^{L} P(q_{ji}(\bar{m}_i)|\bar{R}_j) \quad \forall \quad \bar{m}_i \neq \bar{m}_i' \tag{10}$$

is sufficient for the outer decoder to perform soft-decision MAP decoding under the condition that all message sequences $\bar{m}_i$ are equally likely and all encoded sequence symbols q_{ji} (and hence p_{ij}) are also equally likely. For convolutional codes and TCM codes, the condition that all of the possible output symbols from the outer decoder are equally probable is satisfied.

2.2 The Symbol-by-symbol MAP and "Best-Path" Algorithms

As mentioned previously, there already exists a known algorithm for calculating the a posteriori input probabilities $P(q_i|\bar{R})$. This algorithm is, not surprisingly, known as the symbol-by-symbol MAP algorithm (also known as the "Bahl" [1] or "forward-backward" [7] algorithm). This algorithm in optimum in the sense that it determines the exact values for the a posteriori input probabilities $P(q_i|\bar{R})$. The cost of this precision however is that it requires a relatively large number of multiplications per decoded symbol. This

may pose a problem if the hardware available is not capable of performing the necessary calculations in real-time at the desired speeds[1]. A sub-optimal derivative of this algorithm that reduces the amount of computation required by replacing multiplications with additions and additions with compares is the "Best-Path" algorithm [7]. This algorithm is computationally efficient and capable of delivering high quality probability estimates. However, because of its similarity to the MAP algorithm it also shares some of its disadvantages. Because both of these algorithms involve a forward computational pass starting from the first received symbol and a reverse computational pass starting from the last received symbol, one must wait for the entire transmitted sequence to be received prior to the completion of the calculation of any of the a posteriori input probabilities. This may lead to unacceptable decoding delays at the receiver if the transmitted sequence is relatively long. Furthermore, one must at the very least store the entire received sequence so that it may be referenced for the second pass. Hence, the amount of storage required for this algorithm for long transmitted sequences may be quite large. A detailed description of the MAP and Best-Path algorithms is given in Appendix-C.

2.3 The Soft-Output Viterbi Algorithm (SOVA)

The soft-output Viterbi algorithm is a name used to describe a family of sub-optimal algorithms that are used for calculating estimates of the a posteriori input probabilities $P(q_i|\bar{R})$. They are all based upon a heuristic modification of the conventional Viterbi algorithm (for an explanation of the Viterbi algorithm see [9][10]). Because of this relationship, the decoding delay and the amount of required storage for these algorithms are independent of the length of the transmitted sequence. In addition, the majority

[1]It should be pointed out that many modern microprocessors and digital signal processors are capable of performing floating point operations at speeds comparable to integer arithmetic. Furthermore the cost of this type of hardware appears to be continually decreasing. Therefore, concerns over whether a system has sufficient computing power may no longer be a major issue.

of the computations involved in the calculation of the a posteriori probability estimates involves additions, subtractions, and comparisons – operations that can be performed quite rapidly on simple, inexpensive hardware.

2.3.1 Deducing Viterbi Decision Probabilities

The basic supposition is that it is possible to deduce the reliability of the decisions made by a Viterbi decoder by comparing the path metrics of all paths merging into each state of the trellis. Whenever a Viterbi decoder chooses a survivor merging into a state, the confidence in the input bits or symbols associated with that decision is proportional to the magnitude of the difference between the path metrics of the survivor and concurrent paths. For example, in Figure 2.2 consider the two paths merging into state-1 at decoder memory level $j = 0$. M_s and M_c are respectively the accumulated Viterbi path metrics of the survivor and concurrent paths. If the encoded symbols were transmitted over an AGWN channel, the metrics would be given by:

$$M_i = -\ln P(\text{path-}i \mid r_1^k) = \frac{E_s}{N_o} \sum_{t=1}^{k} |r_t - s_t\{x_t^{(i)}\}|^2 \quad , \qquad i = s, c \tag{11}$$

where r_1^k is the sequence of received symbols from time-1 up to time-k (this notation provides a convenient means of referencing sub-sequences of the entire received sequence $\bar{R}$), r_t is the received sample at time t, and $s_t\{x_t^{(i)}\}$ is the transmitted signal at time t corresponding to the output symbol $x_t^{(i)}$ associated with the i-th path. For this channel, the confidence in the Viterbi decoder's decision in choosing the surviving path over the concurrent path may be found by:

$$\begin{aligned} P\left(\begin{array}{l}\text{Viterbi decision} \\ \text{at } t = k, \text{ state-1}\end{array} \middle| r_1^k \right) &= P(\text{surviving path} | r_1^k) \\ &= \frac{e^{-M_s}}{e^{-M_s} + e^{-M_c}} \\ &= \frac{1}{1 + e^{-(M_c - M_s)}} \end{aligned} \tag{12}$$

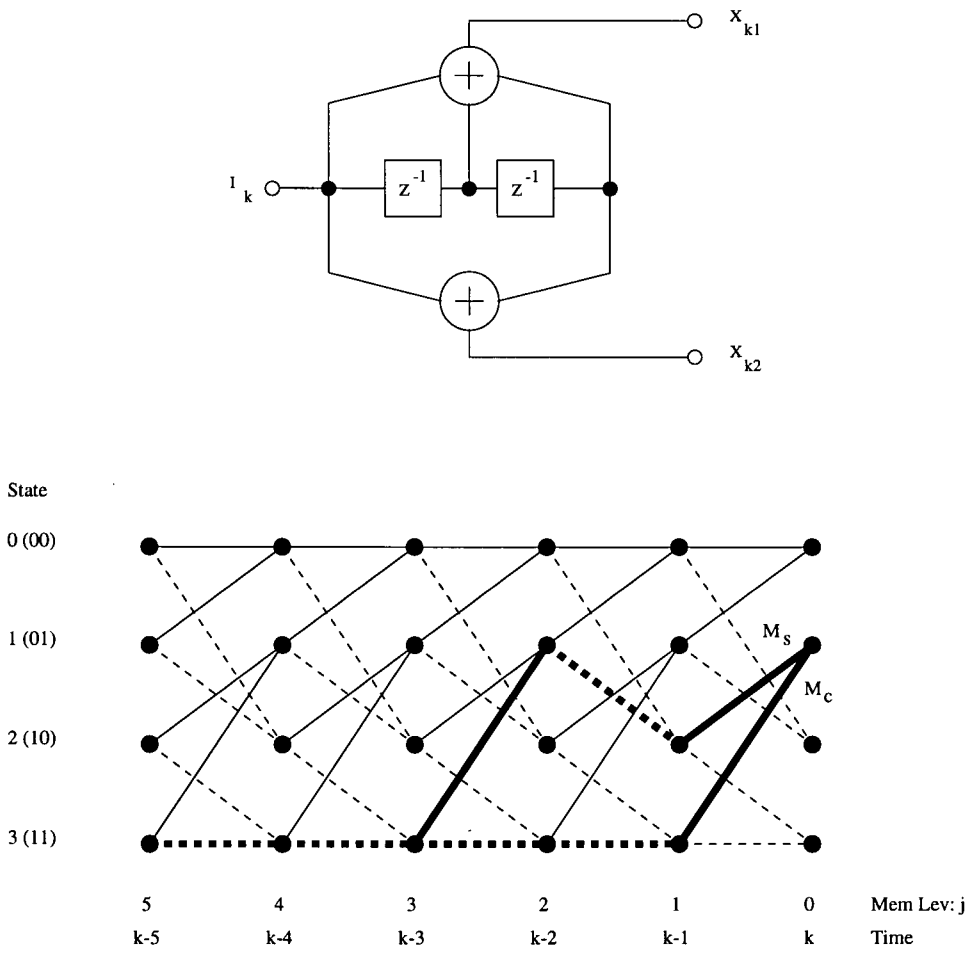


Figure 2.2: "Classic" (2,1,2) Convolutional Encoder and its Corresponding Trellis State Diagram.

Equation (12) provides an expression for determining the probability of the decision made by the Viterbi decoder at time-k and state-1. However, the desired quantity is the probability of the input bits or symbol associated with this selection. The relationship between the decisions and input bits or symbols depends on the structure of the encoder.

Consider the encoder shown in Figure 2.2. For this example, the encoder's state was defined to be the contents of its shift register. By noting the states at time $k-1$ of the survivor and concurrent paths, it becomes apparent that the contents of the encoder's shift register must differ in only the last bit prior to the merger at time-k. Hence the decision probability calculated previously refers the encoder's shift register "roll-off" bit. Equivalently, for this example, the probability calculated corresponds to an input bit decision at time $k-3$.

For feed-forward encoder's in general, each decision associated with the selection of a survivor to each state corresponds to the selection of a corresponding shift-register roll-off symbol. The bits corresponding to this symbol are, of course, bits that would have been inputted previously. Note that if the encoder consists of shift-registers of differing lengths it is not possible to calculate the input symbol probabilities directly.

For a systematic feed-back encoder the relationship is different. In this case, each path merging into a particular state will be associated with unique input symbol. Therefore, the survivor selections also correspond to the selection of a specific input symbol. There is no time delay for this situation.

Taking the previous comments into consideration, for the encoder shown in Figure 2.2, equation (12) states:

$$\begin{aligned} P\left(\begin{array}{c} I_{k-3}=0 \text{ on} \\ \text{surviving path} \\ \text{to state-1} \end{array} \middle| r_1^k \right) &= P\left(\begin{array}{c} \text{Viterbi decision} \\ \text{at } t=k, \text{ state-1} \end{array} \middle| r_1^k \right) \\ &= \frac{1}{1+e^{-(M_c-M_s)}} \end{aligned} \tag{13}$$

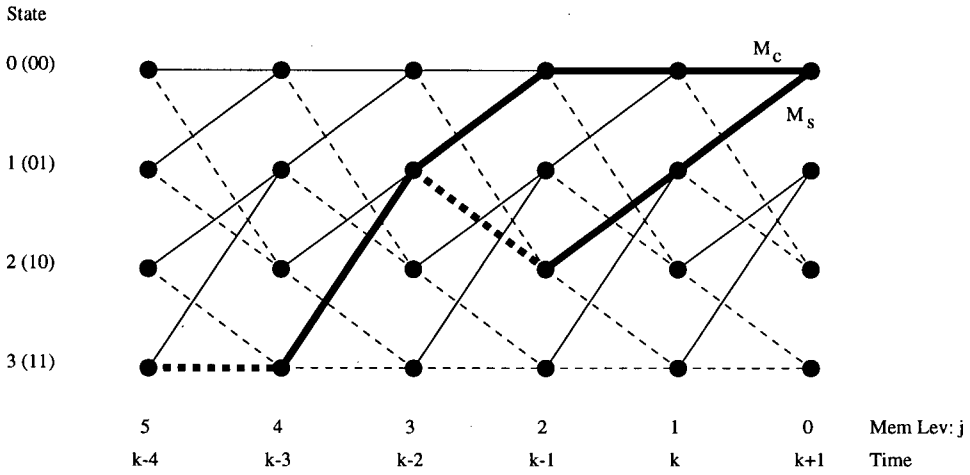


Figure 2.3: Trellis for (2,1,2) Convolutional Code

$$P\left(\begin{array}{c} I_{k-3}=1 \text{ on} \\ \text{surviving path} \\ \text{to state-1} \end{array} \middle| r_1^k\right) = P\left(\begin{array}{c} \text{Concurrent decision} \\ \text{at } t=k, \text{ state-1} \end{array} \middle| r_1^k\right)$$
$$= 1 - P\left(\begin{array}{c} I_{k-3}=0 \text{ on} \\ \text{surviving path} \\ \text{to state-1} \end{array} \middle| r_1^k\right) \quad (14)$$

2.3.2 The Need For Revision

There now exist an expression for the reliability of a Viterbi decision given received symbols $r_1^k = \{r_1, \ldots, r_k\}$. However, because the encoder's output symbols are correlated, as subsequent symbols are received, they will have an impact on the probabilities of previous decisions. The values calculated by equation (12) will have to be updated. Hence one requires a revision algorithm that will update the previously calculated probabilities as new symbols are received. This process of calculating new probabilities and revising the previously calculated probabilities must be repeated until the final symbol r_N arrives. The concept is illustrated in Figure 2.3. Assume that the surviving path from Figure 2.2 is now part of the surviving path to state-0 at time $k+1$. Then if follows that:

$$P\left(\begin{array}{c}I_{k-3}=0 \text{ on}\\ \text{surviving path}\\ \text{to state-0}\end{array}\middle| r_1^{k+1}\right) = P\left(\begin{array}{c}I_{k-3}=0 \text{ on}\\ \text{surviving path}\\ \text{to state-1}\end{array}\middle|\begin{array}{c}\text{Surviving path}\\ \text{at } t=k+1,\\ \text{state-0}\end{array}, r_1^{k+1}\right)\cdot P\left(\begin{array}{c}\text{Surviving path}\\ \text{at } t=k+1,\\ \text{state-0}\end{array}\middle| r_1^{k+1}\right)+$$
$$P\left(\begin{array}{c}I_{k-3}=0 \text{ on}\\ \text{surviving path}\\ \text{to state-0}\end{array}\middle|\begin{array}{c}\text{Concurrent path}\\ \text{at } t=k+1,\\ \text{state-0}\end{array}, r_1^{k+1}\right)\cdot P\left(\begin{array}{c}\text{Concurrent path}\\ \text{at } t=k+1,\\ \text{state-0}\end{array}\middle| r_1^{k+1}\right) \quad (15)$$

Once this revision formula has been applied there will exist up-to-date reliability information for the surviving path for decisions at time $t = k$ and time $t = k + 1$. When the next symbol is received at time $k + 2$, the revision formula is applied once more to the probabilities computed at times k and $k + 1$. The process is repeated until the final symbol is received.

Finding an efficient revision formula or algorithm is the key to SOVA. Where previous works [2][3][6] differ is in the approach and details of this operation. Because of the heavy dependence that the newly proposed revision algorithm has on the previous works, a brief overview of those works shall be presented.

2.4 History of SOVA

2.4.1 Relative Values

In 1987 G. Battail published a paper [2](Fr) describing a general SOVA revision algorithm. Rather than revising the actual Viterbi decision probabilities, as was done in the previous section, he revises a quantity known as "valeur relative" (or "relative values"). For a (n, k, m) code the relative values associated with the decoder's most recent decision are defined as:

$$a_{0\rho} = \log\frac{P(x_0 = 0)}{P(x_0 = \rho)} = M_\rho - M_0 \qquad \rho = 0, \cdots, 2^k - 1 \quad (16)$$

where x_0 represents the encoder's roll-off symbol (for a feed-forward encoder) or input symbol (for a systematic feedback encoder) at the time corresponding to decoder memory depth-0. Equivalently the relative value is representative of the confidence in the Viterbi

decoder's decision concerning the 0^{th} and ρ^{th} paths, as identified by the roll-off or input symbol, to a given state. (This quantity is in-fact the same "relative" metric that was used previously in section 2.3.1.) Using this definition, the confidence in a Viterbi decoder's decision for a non-binary input (n, k, m) convolutional or trellis code is represented by a vector of relative values. Battail also defined the conditional relative value to take into consideration past decisions conditioned on the current choice of survivors to each state. For the decision made j symbols in the past conditioned on the current ρ^{th} path:

$$a^{\rho}_{j\lambda} = \log \frac{P(x_j = 0|x_0 = \rho)}{P(x_j = \lambda|x_0 = \rho)} \qquad \lambda = 0, \cdots, 2^k - 1 \tag{17}$$

With these definitions, Battail derived a general revision formula that was applicable to any (n, k, m) code. However, due to its complexity, he concentrated on the binary input $(n, 1, m)$ case. By focusing on this this sub-class he was able to limit the number of terms that had to be taken into consideration. Simplifying the resulting $(n, 1, m)$ expression he arrived at a revision formula that has a low computational complexity and is relatively simple to implement. His revision formula is reproduced here for the convenience of the reader:

$$a_{(j+1)} = \max(a^1_j, a^0_j + a^1_j, a^0_j + a_0, a^0_j + a^1_j + a_0) - \max(0, a^0_j, a_0, a^1_j + a_0) \tag{18}$$

The reason there is only one equation and not two (one for each $a_{(j+1)i}$, $i = 0, 1$) is that, because of how the relative values are defined, $a_{j0} = 0$. Therefore there is no need to calculate it. Similarly, since $a_{00} = 0$ and $a^i_{j0} = 0$ always, there is no need for those relative values (where they occur in the revision formula, they have been replaced by 0). Therefore, each term in (18) refers to either $a_{(j+1)1}$, a_{j1}, or a^i_{j1}.

To illustrate how this revision formula is used, refer back to Figures 2.2 and 2.3. In Figure 2.2, M_s is the metric for the 0^{th} path (as identified by the shift-register roll-off bit) while M_c is the metric for the 1^{th} path. Therefore, for the decision made at state-1

at time-k:

$$a_{00} = M_s - M_s = 0 \tag{19}$$

$$a_{01} = M_c - M_s \tag{20}$$

Similarly, there would also be a set of relative values generated for the decisions made to each other state. Referring to Figure 2.3 for state-0, time $k+1$, M_c now refers to the 0^{th} path while M_s refers to the 1^{th} path. A new relative value vector $[a_{00}, a_{01}]$ is calculated using these metrics while (19) and (20) become $[a_{10}^1, a_{11}^1]$ since they are now part of the current 1^{th} path. The relative value vector that was calculated for state-0 at time-k now becomes $[a_{10}^0, a_{11}^0]$ as it is part of the current 0^{th} path. Revision formula (18) states that the updated relative value vector for the decision at memory depth-1 (time-k) on the surviving path (the 1^{th} path) to state-0 may be found by:

$$a_{10} = 0 \tag{21}$$

$$a_{11} = \max(a_{01}^1, a_{01}^0 + a_{01}^1, a_{01}^0 + a_{01}, a_{01}^0 + a_{01}^1 + a_{01}) - \max(0, a_{01}^0, a_{01}, a_{01}^1 + a_{01}) \tag{22}$$

2.4.2 A Simplified Revision Algorithm

Presented at Globecom'89, Hagenauer and Hoeher [6] described an alternate revision algorithm. Reproduced in Figure 2.4, the algorithm compares only two quantities per revision and only performs the revision if the concurrent and surviving paths yield different decisions. This algorithm revises the log-likelihood ratio:

$$\hat{L}_j = \log \frac{1 - \hat{p}_j}{\hat{p}_j} \tag{23}$$

where $\hat{p}_j$ is the a posteriori probability estimate for the decision made at memory level j. The revision function $f(\hat{L}_j, \Delta)$ is defined as:

$$f(\hat{L}_j, \Delta) = \frac{1}{\alpha} \log \frac{1 + e^{(\alpha \hat{L}_j + \Delta)}}{e^{\Delta} + e^{\alpha \hat{L}_j}} \approx \min\left(\hat{L}_j, \frac{\Delta}{\alpha}\right) \tag{24}$$

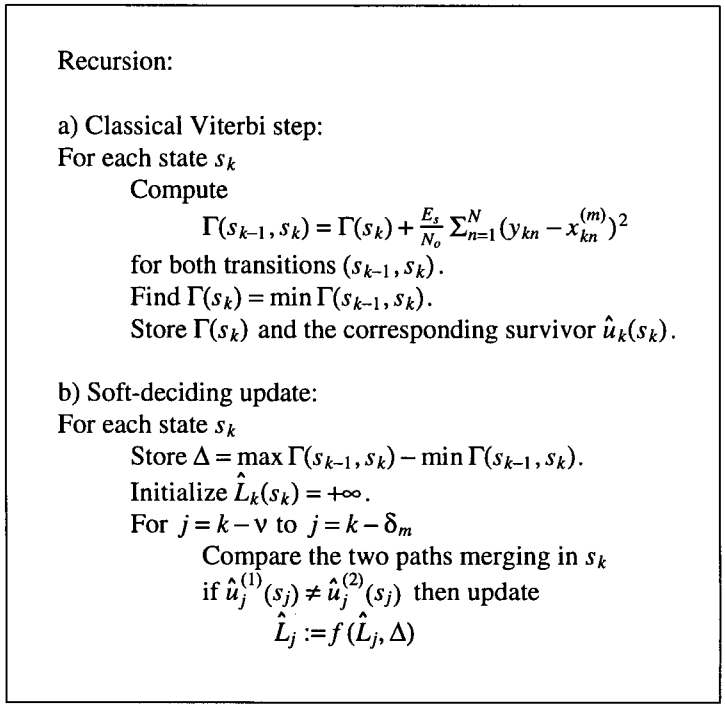


Figure 2.4: Hagenauer and Hoeher's Revision Algorithm

where $\Delta = M_c - M_s$ and α is a constant scaling factor. Like Battail's simplified revision formula, this algorithm is only applicable to $(n, 1, m)$ type codes. However, since this revision formula only compares two quantities per revision, the amount of computation required for this revision algorithm is less.

2.4.3 The Post-Detector Architecture

In a paper presented at ICC'93 by Berrou *et al.* [3] the results of the previous two works are combined. The authors started from Battail's original non-simplified revision formula for $(n, 1, m)$ codes and were able to derive a simple revision algorithm that parallels the one published by Hagenauer and Hoeher in that the revision operation involves essentially the comparison of only two quantities. Like Battail's algorithm their

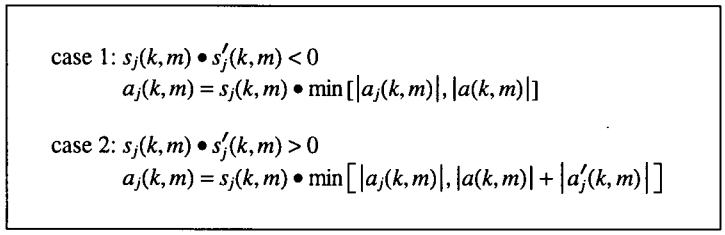


Figure 2.5: Revision Algorithm by Berrou *et al.*

algorithm revises "relative values". Their algorithm is reproduced in Figure 2.5. The $a_j^{(\prime)}(k,m)$ and $a(k,m)$ terms are the same as the a_j^i and a_0 terms in Battail's revision formula (equation (12)) with the exception that the time-k and state-m are explicitly shown. The term $s_j(k,m)$ is the sign of the relative value $a_j(k,m)$. One of the author's findings was that the quality of the a posteriori probability estimates was not seriously affected by the neglecting case-2. This was fortuitous since the author's realized that significant computational savings could be realized by this action. The authors noted that case-1 depends solely on the "relative values" (referred to as "weights" in [3]) of the path currently being revised. Therefore, by choosing to revise only the globally best path (the overall survivor), the "relative value" information of the concurrent paths need neither be computed nor stored. To take advantage of these potential savings, all that is required is that the module performing the SOVA operation know a priori what the surviving path is. This resulted in the post-detector architecture outlined in Figure 2.6.

To illustrate why a post-detector architecture is beneficial for Berrou's algorithm and not Battail's, consider the example shown in Figures 2.2 and 2.3. Consider the hypothetical situation in which both state-1, time-k and state-0, time $k+1$ lie on the global surviving path. Using Battail's notation, according to Berrou *et al.*, the relative value a_{11} can be sufficiently approximated by:

$$a_{11} = \text{sign}(a_{01}^1) \cdot \min[|a_{01}^1|, |a_{01}|] \tag{25}$$

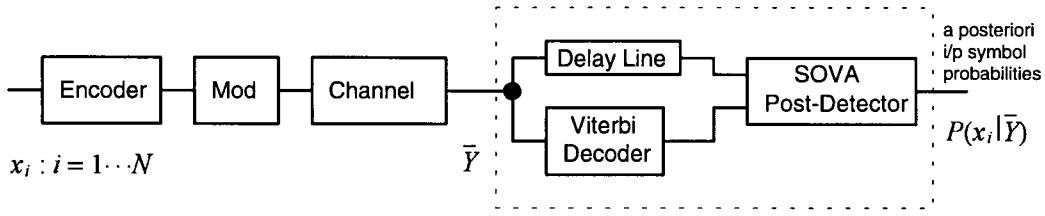


Figure 2.6: Post Detector Architecture

Note that a_{01}^1 is the relative value for state-1, time-k conditioned on the surviving path. Compare this with equation (22) where it was necessary to also know a_{01}^0 - a relative value associated with a decision at a state and time (state-0, time-k) not lying on the globally best path. Therefore, for Battail's algorithm, regardless of having a priori knowledge of what is the globally best path, one still needs have the updated relative values conditioned on the concurrent paths. Since the concurrent paths may have in the past traversed through any trellis state, at each time step one must still revise the relative values of every surviving path to each and every state. As a result, there are no computational or storage savings if a post-detector architecture is used with Battail's algorithm.

At this point, it is convenient to state the conditions under which an algorithm will benefit from a post-detector architecture: The revision algorithm must only depend upon relative values of decisions made on the globally best path. It may depend on the a_{0i} terms if these relative values are for a state lying on the globally best path. It may depend on the a_{ji}^l terms if these relative values are conditioned on being part of the globally best path.

It should be pointed out that the decrease in computational complexity that results from using a post-detector architecture does not come without a price. In this case additional decoding delay has been introduced. Not surprisingly, there appears to be a

trade-off being made between precision, complexity, and decoding delay.

Chapter 3

Derivation of the Generalized Post-Detector

3.1 Introduction

In this chapter the generalized post-detector algorithm is derived. As was done in [3], the derivation presented here begins with Battail's unsimplified revision equation for relative values. However, unlike [3], this treatment begins with the more general equation that is applicable to any (n, k, m) code. For the sake of some of the simplifications that will be made later, and the convenience of the reader, Battail's general revision equation will be re-derived. The derivation parallels the steps as presented in [2], however, the relative values have been referenced against the metric of the surviving path to each state. (As shown in equations (16) and (17), Battail referenced his relative values to the metric of the 0^{th} path to each state.) As shall be shown later, this step makes possible some crucial simplifications that lead to an intermediate revision formula that is essential to the derivation of the final proposed post-detection compatible algorithm.

3.2 Terminology

3.2.1 Relative Values at the Decoder's $0^{\underline{th}}$ memory level

Referring to Figure 3.1 in which two Viterbi decoder m-ary decisions are depicted (one at the 0^{th} memory level and the other at the j^{th} memory level conditioned upon the 0^{th}) the "relative values" of the decoder's most recent decision (at an arbitrary state) are

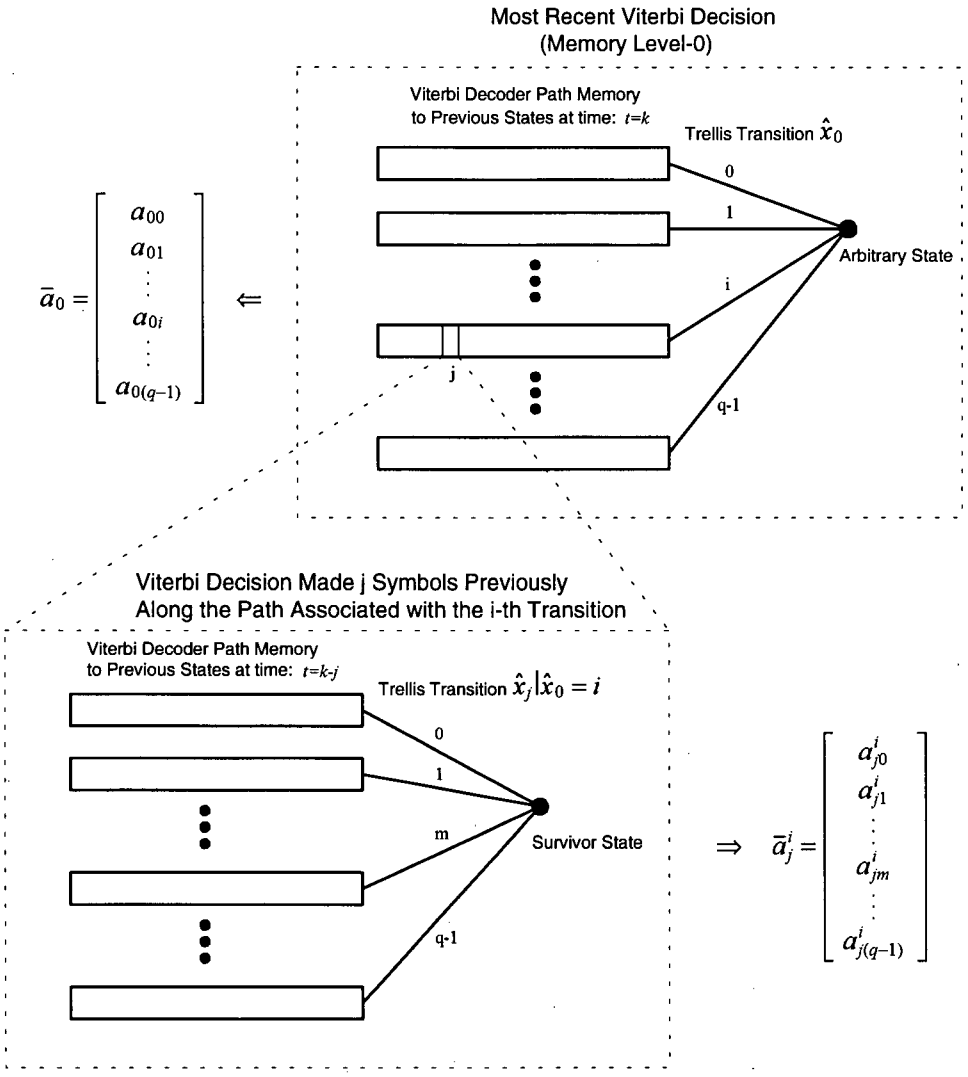


Figure 3.1: Depiction of a Viterbi Decoder's Decisions Made at the $0^{\underline{th}}$ and $j^{\underline{th}}$ Memory Levels

defined as:

$$a_{0m} \triangleq \log \frac{P_r(x_0 = m_s)}{P_r(x_0 = m)}, \quad m = 0, \ldots, q-1, \; m_s \in \{0, \ldots, q-1\} \tag{26}$$

x_0 is a discrete random variable representing the roll-off symbol for a feed-forward encoder or the input symbol for a systematic feedback encoder. Equivalently, it may represent the corresponding path merging into the given state. m represents the possible values for the roll-off or input symbols ranging from 0 to $q-1$. For an (n, k, m) code, q is equal to 2^k. m_s is the specific roll-off or input symbol associated with the surviving path merging into the given state.

For an AWGN channel, if M_m and M_{m_s} are the trellis path metrics of a concurrent path and the survivor path merging into a given state, then the relative values respectively become:

$$a_{0m} = \log \frac{P_r(x_0 = m_s)}{P_r(x_0 = m)} = \log \frac{e^{-M_{m_s}}}{e^{-M_m}} = M_m - M_{m_s} \tag{27}$$

Hence, for each decision made at the decoder's $0^{\underline{th}}$ memory level (one for each possible trellis state), a vector $\bar{a}_0 = [a_{00}, a_{01}, \ldots, a_{0(q-1)}]$ of relative values representing the reliability of that decision is generated. The relative value vector may be viewed as an alternative representation of the p.d.f. of the random variable x_0.

Rearranging (26) yields an expression for the probability of a specific path in terms of its relative value and the probability of the surviving path being traversed:

$$P_r(x_0 = m) = e^{-a_{0m}} P_r(x_0 = m_s) \tag{28}$$

3.2.2 Conditional Relative Values at the Decoder's $j^{\underline{th}}$ memory level

Following a similar procedure as was done for the $0^{\underline{th}}$ memory level, the relative values for decisions made in the past are now defined. Consider a decision made j symbols previously from the current decision conditioned on it being part of the $i^{\underline{th}}$ path to a

specified current state. Define:

$$a_{jm}^{i} \triangleq \log \frac{P_r(x_j = m_s | x_0 = i)}{P_r(x_j = m | x_0 = i)} \ , \quad m = 0, \ldots, q-1 \ , \ m_s \in \{0, \ldots, q-1\} \tag{29}$$

As before the vector $\bar{a}_j^i = [a_{j0}^i, a_{j1}^i, \ldots, a_{j(q-1)}^i]$ is representative of the reliability of the decision made by the decoder.

Rearranging (29) gives an expression for the probability of a specific path traversed j symbols in the past conditioned upon it being part of the current $i^{\underline{th}}$ path:

$$P_r(x_j = m | x_0 = i) = e^{-a_{jm}^i} P_r(x_j = m_s | x_0 = i) \tag{30}$$

For the derivation that follows, it is helpful to fully express $P_r(x_j = m | x_0 = i)$ in terms of relative values. This is readily accomplished by noting that (due to a corollary from the *theorem of total probability)*:

$$\sum_{k=0}^{q-1} P_r(x_j = k | x_0 = i) = 1 \tag{31}$$

By using equation (30):

$$P_r(x_j = m_s | x_0 = i) \sum_{k=0}^{q-1} e^{-a_{jk}^i} = 1 \tag{32}$$

$$P_r(x_j = m_s | x_0 = i) = \frac{1}{\sum_{k=0}^{q-1} e^{-a_{jk}^i}} \tag{33}$$

Now substituting this result back into equation (30) yields:

$$P_r(x_j = m | x_0 = i) = \frac{e^{-a_{jm}^i}}{\sum_{k=0}^{q-1} e^{-a_{jk}^i}} \tag{34}$$

3.3 Re-deriving Battail's Revision Formula

From the *theorem of total probability*:

$$P_r(x_j = i) = \sum_{m=0}^{q-1} P_r(x_j = i | x_0 = m) P_r(x_0 = m) \tag{35}$$

At this point a slight change in notation is made to reflect the fact that as new decisions are made, they are stored in the decoder's path history and all prior decisions are "shifted-up" in memory level by 1. Therefore (35) is rewritten as:

$$P_r(x_{j+1} = i) = \sum_{m=0}^{q-1} P_r(x_j = i | x_0 = m) P_r(x_0 = m) \tag{36}$$

Substitute (28), (34) into equation (36):

$$\begin{aligned} P_r(x_{j+1} = i) &= \sum_{m=0}^{q-1} \left[\frac{e^{-a_{ji}^m}}{\sum_{k=0}^{q-1} e^{-a_{jk}^m}} \right] [e^{-a_{0m}} P_r(x_0 = m_s)] \\ &= \left[\sum_{m=0}^{q-1} \frac{e^{-a_{ji}^m - a_{0m}}}{\sum_{k=0}^{q-1} e^{-a_{jk}^m}} \right] P_r(x_0 = m_s) \end{aligned} \tag{37}$$

With $i = m_s$ (37) yields:

$$P_r(x_{j+1} = m_s) = \left[\sum_{m=0}^{q-1} \frac{e^{-a_{jm_s}^m - a_{0m}}}{\sum_{k=0}^{q-1} e^{-a_{jk}^m}} \right] P_r(x_0 = m_s) \tag{38}$$

Substitute (37) and (38) into:

$$a_{(j+1)i} = \log \frac{P_r(x_{j+1} = m_s)}{P_r(x_{j+1} = i)} \tag{39}$$

yields:

$$a_{(j+1)i} = \log \frac{\left[\sum_{m=0}^{q-1} \frac{e^{-a_{jm_s}^m - a_{0m}}}{\sum_{k=0}^{q-1} e^{-a_{jk}^m}} \right]}{\left[\sum_{m=0}^{q-1} \frac{e^{-a_{ji}^m - a_{0m}}}{\sum_{k=0}^{q-1} e^{-a_{jk}^m}} \right]} \tag{40}$$

Recall that by construction $a_{jm_s}^m \equiv 0, . m = 0, \ldots, q-1$.

$$\boxed{a_{(j+1)i} = \log \left[\sum_{m=0}^{q-1} \frac{e^{-a_{0m}}}{\sum_{k=0}^{q-1} e^{-a_{jk}^m}} \right] - \log \left[\sum_{m=0}^{q-1} \frac{e^{-a_{ji}^m - a_{0m}}}{\sum_{k=0}^{q-1} e^{-a_{jk}^m}} \right]} \tag{41}$$

Equation (41) describes how to "heuristically" revise the relative values of the surviving paths given a Viterbi decoder's most recent decisions. This formula is applicable to any (n, k, m) code however, due to its complexity it is not very practical.

3.4 Simplifying the General Revision Formula

The problems with using equation (41) directly are two fold. First, the outright number of operations required is rather large. Secondly is the fact that many of those operation involve exponentials. To circumvent these problems, it is hoped that many of the terms in equation (41) can be approximated by simpler expressions without drastically affecting the quality of a posteriori input probability estimates. Indeed, this was the primary motivation for re-referencing the relative values to the surviving path since, by doing so, many of the operations can be eliminated.

By using the following approximation to the sum of a set of exponentials:

$$\sum_{k=0}^{q-1} e^{-a_{jk}^m} \approx e^{-min\{a_{jk}^m\}} \tag{42}$$

Due to the re-referencing of the relative values to the surviving path, it follows that for $k \neq m_s$, $a_{jk}^m \geq 0$, while $a_{km_s}^m = 0$. This implies that:

$$\sum_{k=0}^{q-1} e^{-a_{jk}^m} \approx 1 \tag{43}$$

Substitution into equation (41) yields:

$$\begin{aligned} a_{(j+1)i} &\approx \log\left[\frac{\sum_{m=0}^{q-1} e^{-a_{0m}}}{\sum_{m=0}^{q-1} e^{-a_{ji}^m - a_{0m}}}\right] \\ &\approx \log\left[\frac{e^{-min\{a_{0m}\}}}{e^{-min\{a_{ji}^m + a_{0m}\}}}\right] \end{aligned} \tag{44}$$

Similarly since for $m \neq m_s$, $a_{0m} \geq 0$, while $a_{0m_s} = 0$:

$$a_{(j+1)i} \approx \log\left[\frac{1}{e^{-min\{a_{ji}^m + a_{0m}\}}}\right] \tag{45}$$

$$\boxed{a_{(j+1)i} \approx \min_{m=0\ldots q-1}\{a_{ji}^m + a_{0m}\}} \tag{46}$$

3.5 Finding a Revision Algorithm That is Post-Detector Compatible

Equation (46) provides a relatively simple revision formula for any (n, k, m) code. As a soft-output Viterbi decoder selects a survivor merging into a particular state, it can use this formula to revise the stored relative values associated with the surviving path. However, since this operation is done for each surviving path entering into each possible trellis state, the number of revisions required will be quite large. Furthermore, this entails the storage of the relative value vectors associated with each of these surviving paths. Depending upon the hardware available this may not be very practical.

One way to overcome these problems is to try to implement (46) as a "post-detector". The rational behind such a scheme is that if the decoder knows a priori what the globally best path is, it needs only to revise the relative values associated with that path. Why spend precious computational cycles on revising relative values of paths that shall never be emitted by the decoder? Furthermore, if one can eliminate from the revision operation any dependence upon the relative values of the concurrent paths then one only needs to allocate storage for the relative values of a single path. This unfortunately leads to a problem in using equation (46) as it stands for a post-detector. Referring to Figure 3.2, in which a revision operation using equation (46) is depicted, it becomes apparent that as one uses (46) to revise the relative values of a surviving path (at time $t = k + 1$) to a state that is part of the globally best path, one requires knowledge of the relative values of not only the survivor path (to state-b, at time $t = k$) but also of the concurrent paths (to states-a, c, d, at time $t = k$). This implies that at each time step, the decoder must still store and revise the relative values of each survivor to each possible trellis state. This forfeits any potential savings that might have been realized by using a post-detector architecture. If equation (46) can somehow be modified in such a manner so that the revision of the surviving path (at time $t = k + 1$) does not depend on the knowledge of

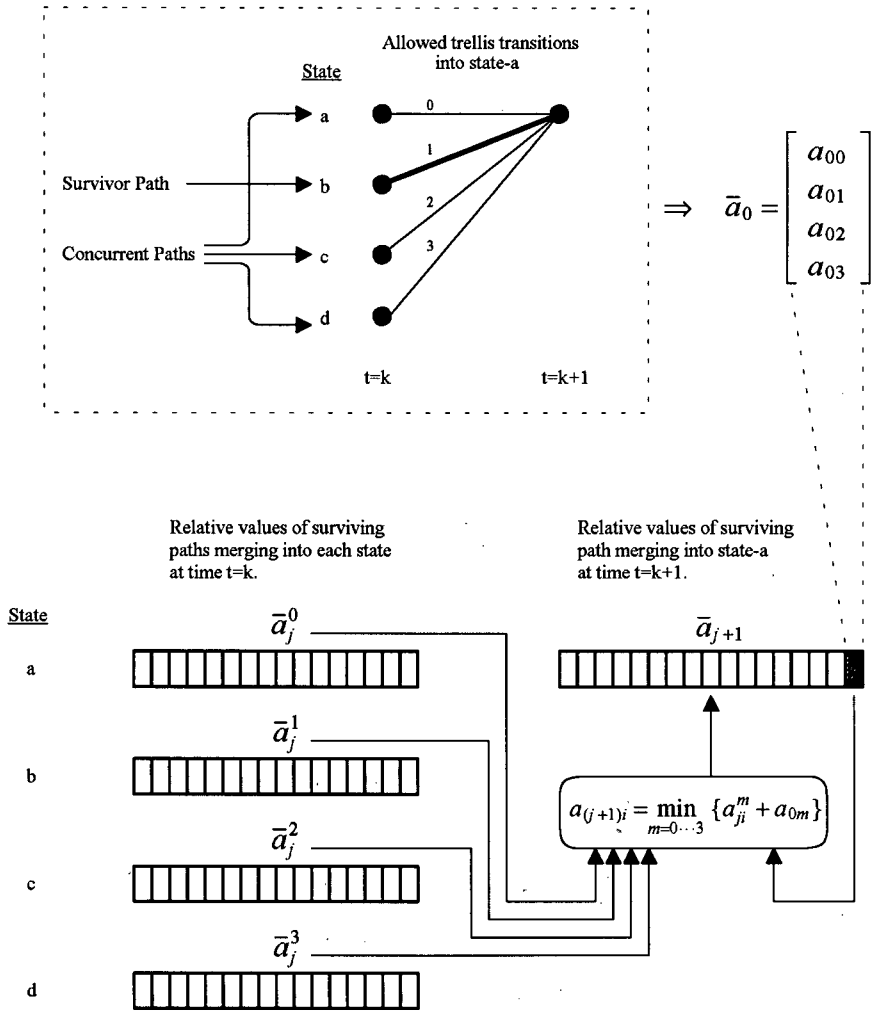


Figure 3.2: Revision Operation Using Equation (46)

concurrent paths (at time $t = k$) then the computational/storage savings hoped to be gained by using a post-detector architecture can be realized.

Insight into finding the appropriate modifications can be gained by considering (46) for the $(n, 1, m)$ case and determining what necessary changes must be made in order that it meet the "post-detector" requirements outlined previously. It is hoped that these modifications will provide sufficient clues on how to adapt (46) into a "post-detector" compatible algorithm that may be applied to any (n, k, m) code.

3.5.1 Post-Detection Compatible Revision Algorithm for $(n, 1, m)$ Codes

From equation (46):

$$a_{(j+1)0} = \min\{a^0_{j0} + a_{00}\,,\; a^1_{j0} + a_{01}\} \tag{47}$$

$$a_{(j+1)1} = \min\{a^0_{j1} + a_{00}\,,\; a^1_{j1} + a_{01}\} \tag{48}$$

There are eight possible cases to consider: $\hat{x}_0 = 0$ or 1, $\hat{x}_j = 0$ or $1|\hat{x}_0 = 0$, and $\hat{x}_j = 0$ or $1|\hat{x}_0 = 1$, where $\hat{x}$ denotes the decision made by a Viterbi decoder. The effect of these Viterbi decisions on the revision equations is shown in the following tables.

Case 1 - (n,1,m) codes		
Mem Lev	Decision	Implied Rel Val's
0	$\hat{x}_0 = 0$	$a_{00} = 0$
j	$\hat{x}_j = 0 \mid \hat{x}_0 = 0$	$a^0_{j0} = 0$
	$\hat{x}_j = 0 \mid \hat{x}_0 = 1$	$a^1_{j0} = 0$
Revision Equations for Surviving Path		
$a_{(j+1)0} = 0$ $a_{(j+1)1} = \min\{a^0_j,\, a^1_j + a_{01}\}$		

Case 2 - (n,1,m) codes		
Mem Lev	Decision	Implied Rel Val's
0	$\hat{x}_0 = 0$	$a_{00} = 0$
j	$\hat{x}_j = 0 \mid \hat{x}_0 = 0$	$a^0_{j0} = 0$
	$\hat{x}_j = 1 \mid \hat{x}_0 = 1$	$a^1_{j1} = 0$
Revision Equations for Surviving Path		
$a_{(j+1)0} = 0$ $a_{(j+1)1} = \min\{a^0_{j1},\, a_{01}\}$		

Case 3 - (n,1,m) codes

Mem Lev	Decision	Implied Rel Val's
0	$\hat{x}_0 = 0$	$a_{00} = 0$
j	$\hat{x}_j = 1 \vert \hat{x}_0 = 0$	$a^0_{j1} = 0$
	$\hat{x}_j = 0 \vert \hat{x}_0 = 1$	$a^1_{j0} = 0$

Revision Equations for Surviving Path

$$a_{(j+1)0} = \min\{a^0_{j0},\, a_{01}\}$$
$$a_{(j+1)1} = 0$$

Case 4 - (n,1,m) codes

Mem Lev	Decision	Implied Rel Val's
0	$\hat{x}_0 = 0$	$a_{00} = 0$
j	$\hat{x}_j = 1 \vert \hat{x}_0 = 0$	$a^0_{j1} = 0$
	$\hat{x}_j = 1 \vert \hat{x}_0 = 1$	$a^1_{j1} = 0$

Revision Equations for Surviving Path

$$a_{(j+1)0} = \min\{a^0_{j0},\, a^1_{j0} + a_{01}\}$$
$$a_{(j+1)1} = 0$$

Case 5 - (n,1,m) codes

Mem Lev	Decision	Implied Rel Val's
0	$\hat{x}_0 = 1$	$a_{01} = 0$
j	$\hat{x}_j = 0 \vert \hat{x}_0 = 0$	$a^0_{j0} = 0$
	$\hat{x}_j = 0 \vert \hat{x}_0 = 1$	$a^1_{j0} = 0$

Revision Equations for Surviving Path

$$a_{(j+1)0} = 0$$
$$a_{(j+1)1} = \min\{a^0_{j1} + a_{00},\, a^1_{j1}\}$$

Case 6 - (n,1,m) codes

Mem Lev	Decision	Implied Rel Val's
0	$\hat{x}_0 = 1$	$a_{01} = 0$
j	$\hat{x}_j = 0 \vert \hat{x}_0 = 0$	$a^0_{j0} = 0$
	$\hat{x}_j = 1 \vert \hat{x}_0 = 1$	$a^1_{j1} = 0$

Revision Equations for Surviving Path

$$a_{(j+1)0} = \min\{a_{00},\, a^1_{j0}\}$$
$$a_{(j+1)1} = 0$$

Case 7 - (n,1,m) codes

Mem Lev	Decision	Implied Rel Val's
0	$\hat{x}_0 = 1$	$a_{01} = 0$
j	$\hat{x}_j = 1 \vert \hat{x}_0 = 0$	$a^0_{j1} = 0$
	$\hat{x}_j = 0 \vert \hat{x}_0 = 1$	$a^1_{j0} = 0$

Revision Equations for Surviving Path

$$a_{(j+1)0} = 0$$
$$a_{(j+1)1} = \min\{a_{00},\, a^1_{j1}\}$$

Case 8 - (n,1,m) codes

Mem Lev	Decision	Implied Rel Val's
0	$\hat{x}_0 = 1$	$a_{01} = 0$
j	$\hat{x}_j = 1 \vert \hat{x}_0 = 0$	$a^0_{j1} = 0$
	$\hat{x}_j = 1 \vert \hat{x}_0 = 1$	$a^1_{j1} = 0$

Revision Equations for Surviving Path

$$a_{(j+1)0} = \min\{a^0_{j0} + a_{00},\, a^1_{j0}\}$$
$$a_{(j+1)1} = 0$$

To find an algorithm that is compatible with the post-detection scheme presented in [3] it is necessary that revision depend only on the relative values of the surviving path and the most recent decision. This is true for cases: 2, 3, 6, 7 – when the decisions made at memory level-j were different.

To uncover what additional approximations have to be made to make the other cases (when the decisions made at memory level-j were identical) post-detector compatible, consider case-1. According to the algorithms by Berrou *et al.* and Hagenaurer/Hoeher, revision is not necessary if both the concurrent and survivor paths yield the same decision. Therefore, in compliance with their findings, revision should not be necessary for case-1. Expressed in the notation used for this chapter this would imply:

$$\begin{aligned} a_{(j+1)0} &= a^0_{j0} \\ a_{(j+1)1} &= a^0_{j1} \end{aligned} \tag{49}$$

In case-1, the revision equation for $a_{(j+1)0}$ satisfies (49) already since $a^0_{j0} = 0$. The revision equation for $a_{(j+1)1}$ can be made to satisfy (49) if the $a^1_{j1} + a_{01}$ term is neglected. This is not entirely unreasonable since by construction all of the terms are positive. In essence, by neglecting this term, the assumption made is that the sum of two terms will always be greater than a single term. Examination of cases 4, 5, and 8 show that they too can be made post-detector compatible by making this same assumption. The resulting revision equations follow:

Case 1	Case 2
$a_{(j+1)0} = a^0_{j0}$ $a_{(j+1)1} = a^0_{j1}$	$a_{(j+1)0} = a^0_{j0}$ $a_{(j+1)1} = \min\{a^0_{j1}, a_{01}\}$
Case 3	**Case 4**
$a_{(j+1)0} = \min\{a^0_{j0}, a_{01}\}$ $a_{(j+1)1} = a^0_{j1}$	$a_{(j+1)0} = a^0_{j0}$ $a_{(j+1)1} = a^0_{j1}$
Case 5	**Case 6**
$a_{(j+1)0} = a^1_{j0}$ $a_{(j+1)1} = a^1_{j1}$	$a_{(j+1)0} = \min\{a_{00}, a^1_{j0}\}$ $a_{(j+1)1} = a^1_{j1}$
Case 7	**Case 8**
$a_{(j+1)0} = a^1_{j0}$ $a_{(j+1)1} = \min\{a_{00}, a^1_{j1}\}$	$a_{(j+1)0} = a^1_{j0}$ $a_{(j+1)1} = a^1_{j1}$

This can be shown to be the exact same algorithm, given notational differences, as the one proposed by Berrou *et al.* in [3] for their post-detector.

To summarize, it was discovered that by using the revision equation (46) for binary $(n, 1, m)$ codes, and by applying the assumption that the sum of two terms is always greater than a single term, it was possible to derive a revision algorithm (or set of equations in this case) that was post-detector compatible. It now remains to be shown that by using the same equation for the more general case of (n, k, m) codes and by applying the same assumption, this will also result in a post-detector compatible revision algorithm.

3.5.2 Post-Detection Compatible Revision Algorithm for (n, k, m) Codes

To derive a revision algorithm for the non-binary (n, k, m) case a procedure similar to that for the binary $(n, 1, m)$ case is followed. The case where $k = 2$ is used to illustrate the effects of the assumptions made. This will keep the number of terms that must be considered in the case studies reasonable. Since none of the assumptions made will be specific to the case where $k = 2$ the resulting algorithm should be applicable to the more general case.

For an $(n, 2, m)$ code, equation (46) yields:

$$\begin{aligned} a_{(j+1)0} &= \min\{a^0_{j0} + a_{00}\,,\, a^1_{j0} + a_{01}\,,\, a^2_{j0} + a_{02}\,,\, a^3_{j0} + a_{03}\} \\ a_{(j+1)1} &= \min\{a^0_{j1} + a_{00}\,,\, a^1_{j1} + a_{01}\,,\, a^2_{j1} + a_{02}\,,\, a^3_{j1} + a_{03}\} \\ a_{(j+1)2} &= \min\{a^0_{j2} + a_{00}\,,\, a^1_{j2} + a_{01}\,,\, a^2_{j2} + a_{02}\,,\, a^3_{j2} + a_{03}\} \\ a_{(j+1)3} &= \min\{a^0_{j3} + a_{00}\,,\, a^1_{j3} + a_{01}\,,\, a^2_{j3} + a_{02}\,,\, a^3_{j3} + a_{03}\} \end{aligned} \tag{50}$$

Is Revision Necessary if All Antecedents Yield The Same Decision?

Although, with increasing k, this scenario becomes less likely early on in the Viterbi decision process, it is still applicable once all of the surviving paths have merged at some arbitrary memory depth ([12] implies that this usually occurs within 5 constraint lengths). This prevents the decoder from performing revision over the entire decoder memory. It will only be done where it is "necessary" to achieve satisfactory a posteriori information.

Consider the following case studies:

Case 1 - (n,2,m) codes

Mem Lev	Decision	Resulting Known Rel Val
0	$\hat{x}_0 = 0$	$a_{00} = 0$
j	$\hat{x}_j = 0 \mid \hat{x}_0 = 0$	$a^0_{j0} = 0$
	$\hat{x}_j = 0 \mid \hat{x}_0 = 1$	$a^1_{j0} = 0$
	$\hat{x}_j = 0 \mid \hat{x}_0 = 2$	$a^2_{j0} = 0$
	$\hat{x}_j = 0 \mid \hat{x}_0 = 3$	$a^3_{j0} = 0$

Resulting Revision Equations for Surviving Path	Observations
$a_{(j+1)0} = 0$	$= a^0_{j0}$
$a_{(j+1)1} = \min\{a^0_{j1}\,,\, a^1_{j1} + a_{01}\,,\, a^2_{j1} + a_{02}\,,\, a^3_{j1} + a_{03}\}$	$\approx a^0_{j1}$
$a_{(j+1)2} = \min\{a^0_{j2}\,,\, a^1_{j2} + a_{01}\,,\, a^2_{j2} + a_{02}\,,\, a^3_{j2} + a_{03}\}$	$\approx a^0_{j2}$
$a_{(j+1)3} = \min\{a^0_{j3}\,,\, a^1_{j3} + a_{01}\,,\, a^2_{j3} + a_{02}\,,\, a^3_{j3} + a_{03}\}$	$\approx a^0_{j3}$

Case 2 - (n,2,m) codes

Mem Lev	Decision	Resulting Known Rel Val
0	$\hat{x}_0 = 0$	$a_{00} = 0$
j	$\hat{x}_j = 1 \mid \hat{x}_0 = 0$	$a^0_{j1} = 0$
	$\hat{x}_j = 1 \mid \hat{x}_0 = 1$	$a^1_{j1} = 0$
	$\hat{x}_j = 1 \mid \hat{x}_0 = 2$	$a^2_{j1} = 0$
	$\hat{x}_j = 1 \mid \hat{x}_0 = 3$	$a^3_{j1} = 0$

Resulting Revision Equations for Surviving Path	Observations
$a_{(j+1)0} = \min\{a^0_{j0}\,,\, a^1_{j0} + a_{01}\,,\, a^2_{j0} + a_{02}\,,\, a^3_{j0} + a_{03}\}$	$\approx a^0_{j0}$
$a_{(j+1)1} = 0$	$= a^0_{j1}$
$a_{(j+1)2} = \min\{a^0_{j2}\,,\, a^1_{j2} + a_{01}\,,\, a^2_{j2} + a_{02}\,,\, a^3_{j2} + a_{03}\}$	$\approx a^0_{j2}$
$a_{(j+1)3} = \min\{a^0_{j3}\,,\, a^1_{j3} + a_{01}\,,\, a^2_{j3} + a_{02}\,,\, a^3_{j3} + a_{03}\}$	$\approx a^0_{j3}$

From the previous two examples it can be seen that if the assumption that the sum of two relative value terms is always greater than a single relative value term then revision is not necessary when all of the "antecedents" (surviving paths to each allowed state at the previous time step) yield the same decision. Stated in another way, once all of the surviving paths to each state in the trellis have merged, say at memory level-j', then no revision is necessary beyond this point.

Revision Algorithm for Case When All Paths Do Not Yield The Same Decision

To determine the revision algorithm when the antecedents do not all yield the same decision, it is convenient to place all of the terms of equation (50) in a revision matrix:

$$\begin{bmatrix} a^0_{j0}+a_{00} & a^1_{j0}+a_{01} & a^2_{j0}+a_{02} & a^3_{j0}+a_{03} \\ a^0_{j1}+a_{00} & a^1_{j1}+a_{01} & a^2_{j1}+a_{02} & a^3_{j1}+a_{03} \\ a^0_{j2}+a_{00} & a^1_{j2}+a_{01} & a^2_{j2}+a_{02} & a^3_{j2}+a_{03} \\ a^0_{j3}+a_{00} & a^1_{j3}+a_{01} & a^2_{j3}+a_{02} & a^3_{j3}+a_{03} \end{bmatrix} \tag{51}$$

Suppose the most recent decision to a given state is $\hat{x}_0 = i$, then $a_{0i} = 0$. This leaves only the a^i_{jk} terms in column i+1 of matrix (51). Since, in the resulting revision algorithm, the assumption is made that the sum of two relative values will always be greater than any single relative value, this implies the dependency of the resulting algorithm on the relative value terms of the surviving path (at time step $t = k$).

Now consider the case where the $n^{\underline{th}}$ path yields the following decision: $\hat{x}_j = l|\hat{x}_0 = n$. This leads to $a^n_{jl} = 0$ which eliminates a term in the $l + 1^{th}$ row and $n + 1^{th}$ column leaving only the a_{0n} term. Therefore, the decisions made on the preceding paths at the $j^{\underline{th}}$ level indicate which relative value terms a_{0n} are relevant in making the current revision. The following example illustrates this finding:

Example - (n,2,m) codes		
Mem Lev	Decision	Known Rel Val
0	$\hat{x}_0 = 1$	$a_{01} = 0$
j	$\hat{x}_j = 3 \mid \hat{x}_0 = 0$	$a^0_{j3} = 0$
	$\hat{x}_j = 0 \mid \hat{x}_0 = 1$	$a^1_{j0} = 0$
	$\hat{x}_j = 2 \mid \hat{x}_0 = 2$	$a^2_{j2} = 0$
	$\hat{x}_j = 2 \mid \hat{x}_0 = 3$	$a^3_{j2} = 0$

Resulting Revision Equations for Surviving Path	Post-Detector Revision Equations
$a_{(j+1)0} = \min\{a^0_{j0} + a_{00}\,,\, 0+0\,,\, a^2_{j0} + a_{02}\,,\, a^3_{j0} + a_{03}\}$	$= \min\{a^1_{j0}\,,\, a_{01}\}$
$a_{(j+1)1} = \min\{a^0_{j1} + a_{00}\,,\, a^1_{j1} + 0\,,\, a^2_{j1} + a_{02}\,,\, a^3_{j1} + a_{03}\}$	$\approx a^1_{j1}$
$a_{(j+1)2} = \min\{a^0_{j2} + a_{00}\,,\, a^1_{j2} + 0\,,\, 0 + a_{02}\,,\, 0 + a_{03}\}$	$\approx \min\{a^1_{j2}\,,\, a_{02}\,,\, a_{03}\}$
$a_{(j+1)3} = \min\{0 + a_{00}\,,\, a^1_{j3} + 0\,,\, a^2_{j3} + a_{02}\,,\, a^3_{j3} + a_{03}\}$	$\approx \min\{a^1_{j3}\,,\, a_{00}\}$

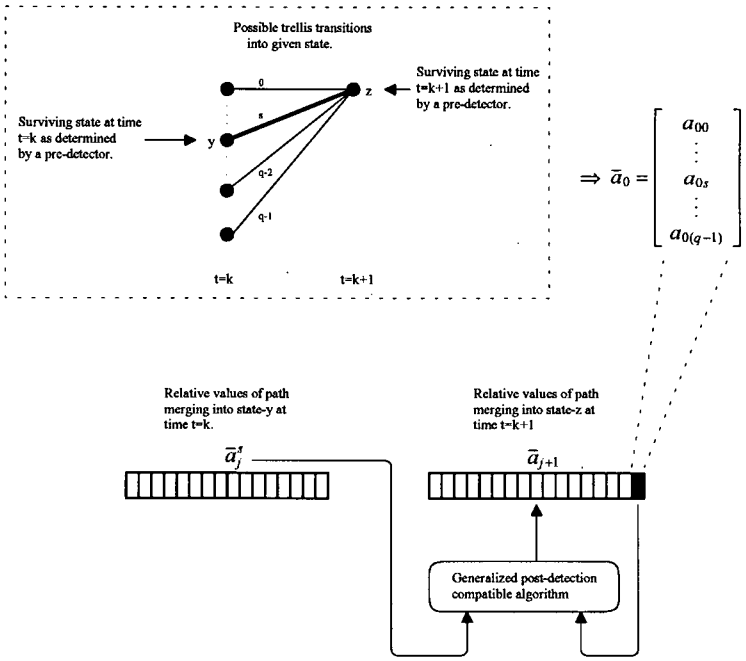
Note that the approximate revision algorithm only requires knowledge of the surviving path's relative values $\{a^s_{j0}\,,\, a^s_{j1}\,,\, a^s_{j2}\,,\, a^s_{j3}\}$ and the relative values corresponding to the most recent decision $\{a_{00}\,,\, a_{01}\,,\, a_{02}\,,\, a_{03}\}$. Hence this algorithm appears to be the M-ary equivalent of the algorithm presented by Berrou *et al.* in [3].

By comparing the decisions made by the decoder with the resulting revision equations a pattern becomes apparent. This pattern leads to the revision algorithm outlined in Figure 3.3.

3.6 Application Considerations

3.6.1 Deducing Input Probabilities From Relative Values

To this point consideration has only been given to the problem of finding an efficient method of revising the relative values derived from a Viterbi decoder's decisions. As mentioned previously, to deduce the a posteriori probabilities of the encoder's input bits



To update $a_{(j+1)i}$ given that $\hat{x}_0 = s$ (ie $a_{0s} = 0$),

If all paths merging into the current state yield the same decision at memory depth j then:

No revision is necessary.
Set $a_{(j+1)i} = a_{ji}^s$.

else

Are any of the decisions made, at memory depth j, on the possible merging paths equal to index i ?

No → No revision is necessary.
Set $a_{(j+1)i} = a_{ji}^s$.

Yes → Say on paths m_1 and m_2. Revision is required.
Set $a_{(j+1)i} = \min\{a_{ji}^s, a_{0m_1}, a_{0m_2}\}$.

(end).

Figure 3.3: Generalized Revision Algorithm for Post-Detector

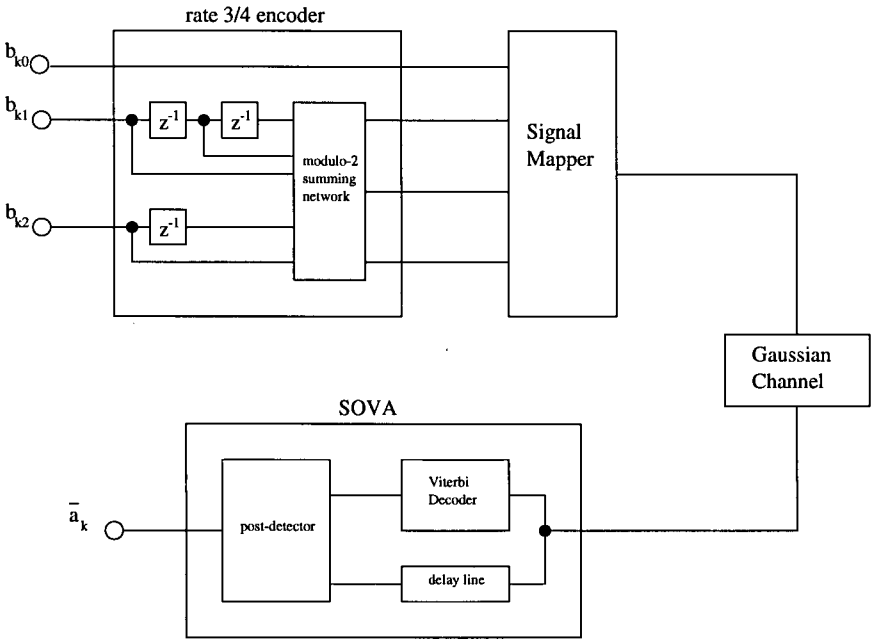


Figure 3.4: Rate 3/4 SOVA system

or symbols from the relative values, one must consider the structure of the encoder. To illustrate how this is accomplished consider the following two examples.

Example 1: Feed-Forward Encoder

Consider finding the a posteriori input bit probabilities for the rate-$\frac{3}{4}$ system depicted in Figure 3.4. Suppose the SOVA outputs the following relative value vector at time $t = k$:

$$\bar{a}_k = [a_{k0}, a_{k1}, a_{k2}, a_{k3}, a_{k4}, a_{k5}, a_{k6}, a_{k7}]$$

If the paths merging into each state were identified using the shift-register "roll-off" symbol then:

$$a_{k0} = \log \frac{P(x_k = 0)}{P(x_k = m_s)} = \log \frac{P(b_{k0} = 0,\ b_{(k-2)1} = 0,\ b_{(k-1)2} = 0)}{P(b_{k0} = m_{s0},\ b_{(k-2)1} = m_{s1},\ b_{(k-1)2} = m_{s2})}$$

Therefore,

$$P(b_{k0}=0) = \frac{\sum_{l|b_{k0}=0} P(x_k=l)}{\sum_{l=0}^{7} P(x_k=l)} = \frac{\sum_{l|b_{k0}=0} \frac{P(x_k=l)}{P(x_k=m_s)}}{\sum_{l=0}^{7} \frac{P(x_k=l)}{P(x_k=m_s)}}$$

$$= \frac{\sum_{l|b_{k0}=0} e^{-a_{kl}}}{\sum_{l=0}^{7} e^{-a_{kl}}}$$

$$P(b_{k0}=1) = 1 - P(b_{k0}=0)$$

Similarly it is possible to deduce $P(b_{(k-2)1}=0)$ and $P(b_{(k-1)2}=0)$. To determine $P(b_{k1}=0)$ and $P(b_{k2}=0)$ one must use the relative value vectors $\bar{a}_{k+2}$ and $\bar{a}_{k+1}$ respectively. This implies that one must buffer several relative values vectors after the completion of revision or use relative value vectors that may not have been fully revised. The latter situation should be acceptable so long as the revision memory is sufficiently long such that all concurrent paths have merged for D symbols prior to symbol output – where D is the maximum length shift register. Note that it is not possible to determine the a posteriori input symbol probabilities for this encoder since the shift-registers are of differing lengths.

Example 2: ISI Channel

Consider finding the a posteriori input symbol and bit probabilities for the ISI channel shown in Figure 3.5. Because the signaling constellation is 8-PSK, the relative value vector will have 8 components:

$$\bar{a}_k = [a_{k0}, a_{k1}, a_{k2}, a_{k3}, a_{k4}, a_{k5}, a_{k6}, a_{k7}]$$

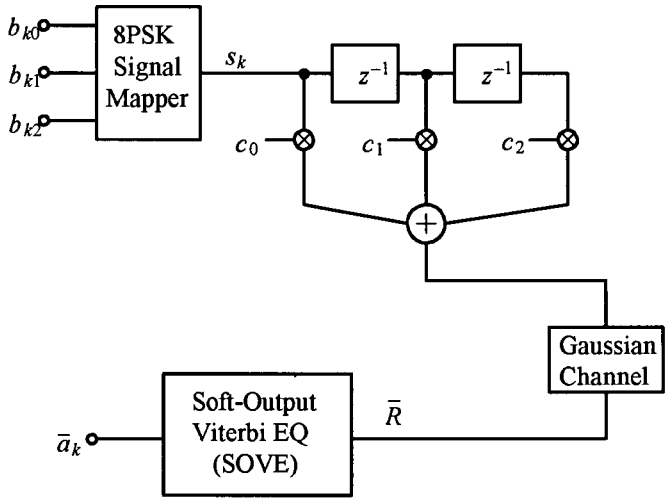


Figure 3.5: Rate-3 Soft-Output Viterbi EQ system

For this case the shift-register "roll-off" symbol is just the ISI channel input symbol offset by two time periods. Therefore given that:

$$a_{k0} = \log \frac{P(x_k = 0)}{P(x_k = m_s)} = \log \frac{P(s_{k-2} = 0)}{P(b_{k-2} = m_s)}$$

the 0-symbol probability may be found by:

$$P(s_{k-2} = 0) = \frac{P(s_{k-2} = 0)}{\sum_{l=0}^{7} P(s_{k-2} = l)} = \frac{\frac{P(s_{k-2}=0)}{P(s_{k-2}=m_s)}}{\sum_{l=0}^{7} \frac{P(s_{k-2}=l)}{P(s_{k-2}=m_s)}} = \frac{e^{-a_{k0}}}{\sum_{l=0}^{7} e^{-a_{kl}}}$$

The probabilities for the other symbols can be calculated similarly.

Determining the a posteriori input bit probabilities probabilities is a fairly straight forward procedure given knowledge of the mapping used by the modulator. They can either be calculated directly as was done in Example-1 or calculated from the a posteriori input symbol probabilities. For these simulations the mapping used is shown in Table 3.1.

$b_0b_1b_2$	$\rightarrow$	$s_n : \alpha_n + j\,\beta_n$
000	$\rightarrow$	$0 : +\cos(22.5°) - j\sin(22.5°)$
001	$\rightarrow$	$1 : -\sin(22.5°) + j\cos(22.5°)$
010	$\rightarrow$	$2 : +\cos(22.5°) + j\sin(22.5°)$
011	$\rightarrow$	$3 : +\sin(22.5°) + j\cos(22.5°)$
100	$\rightarrow$	$4 : -\cos(22.5°) + j\sin(22.5°)$
101	$\rightarrow$	$5 : +\sin(22.5°) - j\cos(22.5°)$
110	$\rightarrow$	$6 : -\cos(22.5°) - j\sin(22.5°)$
111	$\rightarrow$	$7 : -\sin(22.5°) - j\cos(22.5°)$

Table 3.1: Signal Mapping used by the 8-PSK Modulator.

Using the a posteriori input symbol probabilities, $P(b_{(k-2)0})$ may be found by:

$$\begin{aligned} P(b_{(k-2)0} = 0) &= P(s_{k-2} = 0) + P(s_{k-2} = 1) + P(s_{k-2} = 2) + P(s_{k-2} = 3) \\ P(b_{(k-2)0} = 1) &= 1 - P(b_{k-2} = 0) \end{aligned}$$

The probabilities for $P(b_{(k-2)1})$ and $P(b_{(k-2)2})$ can be found in a similar manner.

3.6.2 Examining the Effects of the Approximations Made

During the derivation of the generalized post-detection algorithm several approximations were made to arrive at a simple expression. It may be interesting to determine how these approximations affected the "physical" meaning of the equations.

The unsimplified revision equation (equation (40)) can be rearranged as follows:

$$e^{a(j+1)i} = \frac{P_r(\hat{x}_{j+1} = m_s)}{P_r(\hat{x}_{j+1} = i)} = \frac{\left[\sum_{m=0}^{q-1} \frac{e^{-a^m_{jm_s} - a_{0m}}}{\sum_{k=0}^{q-1} e^{-a^m_{jk}}}\right] P_r(\hat{x}_0 = m_s)}{\left[\sum_{m=0}^{q-1} \frac{e^{-a^m_{ji} - a_{0m}}}{\sum_{k=0}^{q-1} e^{-a^m_{jk}}}\right] P_r(\hat{x}_0 = m_s)} \tag{52}$$

For the time being consider only the denominator terms.

$$P_r(\hat{x}_{j+1} = i) = \sum_{m=0}^{q-1} [e^{-a^m_{ji}}] \left[\frac{1}{\sum_{k=0}^{q-1} e^{-a^m_{jk}}}\right] [e^{-a_{0m}} P_r(\hat{x}_0 = m_s)] \tag{53}$$

$$= \sum_{m=0}^{q-1} \underbrace{\left[\frac{P_r(\hat{x}_j = i|\hat{x}_0 = m)}{P_r(\hat{x}_j = m_s|\hat{x}_0 = m)}\right]}_{\alpha} \underbrace{[P_r(\hat{x}_j = m_s|\hat{x}_0 = m)]}_{\beta} \underbrace{[P_r(\hat{x}_0 = m)]}_{\gamma}$$

The first approximation made during the derivations was as follows:

$$\beta = \left[\frac{1}{\sum_{k=0}^{q-1} e^{-a_{jk}^m}}\right] \approx \left[\frac{1}{e^{-a_{jk'}^m}}\right], \text{ where } a_{jk'}^m \leq a_{jk}^m \, \forall \, k, \; k \neq k' \tag{54}$$

Since by construction the survivor's relative value $a_{jk'}^m = 0$:

$$P_r(\hat{x}_j = m_s|\hat{x}_0 = m) \approx 1 \tag{55}$$

This implies that given knowledge of the encoder's most recent decision, the decoder is certain of the encoder's decision j symbols previously. Stated another way, the decoder assumes it made the correct decision j symbols in the past.

Applying this same assumption to the numerator as well the denominator leads to the following equation:

$$e^{a_{(j+1)i}} = \frac{P_r(\hat{x}_{j+1} = m_s)}{P_r(\hat{x}_{j+1} = i)} = \frac{\sum_{m=0}^{q-1} [e^{-a_{0m}} P_r(\hat{x}_0 = m_s)]}{\sum_{m=0}^{q-1} [e^{-a_{ji}^m}][e^{-a_{0m}} P_r(\hat{x}_0 = m_s)]} \tag{56}$$

The next approximation made involved the numerator of this equation.

$$\sum_{m=0}^{q-1} e^{-a_{0m}} \approx e^{-a_{0m'}}, \text{ where } a_{0m'} \leq a_{0m} \, \forall \, m, \; m \neq m' \tag{57}$$

Once again, by construction, $a_{0m'} = 0$. Therefore,

$$\sum_{m=0}^{q-1} \frac{P_r(\hat{x}_0 = m)}{P_r(\hat{x}_0 = m_s)} \approx 1 \tag{58}$$

$$\sum_{m=0}^{q-1} P_r(\hat{x}_0 = m) \approx P_r(\hat{x}_0 = m_s) \tag{59}$$

This implies that:

$$P_r(x_0 = m_s) \gg P_r(x_0 = m) \; , \; \forall \, m, \neq m_s \tag{60}$$

Determining the physical meaning of the approximations made beyond this point becomes quite difficult. However, the two assumptions that have been uncovered (equations (55) and (60)) do provide some indication of what conditions are required for the derived algorithm to work well. The two assumptions are generally true for codes with good distance qualities and high SNR. Since most system designers generally choose such codes and operate at moderate signal strengths these conditions should not pose any problems for systems such as concatenated coding where the inner code is a convolutional code or a TCM code. However, these conditions also suggest that the new revision algorithm may not be suited for applications such as Viterbi equalization where the channel may not necessarily conform to a "good code". To determine if this is true, computer simulations would have to be performed.

3.6.3 Computational and Storage Requirements

Because of the generalized post-detector's dependency upon the Viterbi algorithm, the computational and storage requirements of the standard Viterbi decoder shall be reviewed. The additional resources required to implement the generalized post-detector shall then be discussed. The results will be compared with the resource requirements of the MAP algorithm and the Best-Path algorithm.

Consider a typical (n, k, m) convolutional code. The trellis diagram of this code will have 2^m states. If δ_{vtb} represents the length of the past history memory of the corresponding Viterbi decoder then $\delta_{vtb} \cdot 2^m$ storage elements (most likely integer quantities) for the path history arrays will need to be allocated. In addition, 2^m additional storage elements (typically floating point quantities) are required to hold the metrics of each of the surviving paths.

The Viterbi decoding process involves for each received symbol: 2^n branch metric calculations followed by $2^m \cdot 2^k$ additions for path metric calculations. In addition, 2^m

comparisons of 2^k path metrics are required for the selection of the surviving paths to each state. If it is assumed that a comparison of 2^k items can be accomplished by $2^k - 1$ binary comparisons then that leads to a requirement of $2^m(2^k - 1)$ binary comparisons per received symbol. These results are summarized in Table 3.2. The exact number of additions/multiplications for the branch metric calculation will not be considered as it depends on the specific application - most notably, the type of modulation used. However, since all of the decoding methods to be considered here require the computation of a similar quantity, it can be considered as a constant computational overhead that is not a factor when selecting between the various algorithms.

The addition of the generalized post-detector derived in Chapter-3 will entail a second Viterbi decoder to calculate the path metrics for the relative value calculations. If the post-detector has a path memory of length δ_{pd} the this will require an additional $\delta_{pd} \cdot 2^m$ storage elements for path histories to each state. As in the standard Viterbi decoder each storage element must store the input symbol decision. However, if a feed-forward code is used, the roll-off symbol will also have to be stored. Hence the required storage for the path history arrays is $2\delta_{pd} \cdot 2^m$ (probably integer quantities). As before, 2^m elements for path metric storage (floating point quantities) are required. The relative value arrays require $\delta_{pd} \cdot 2^k$ elements (floating point).

Viterbi Storage Requirements	
Integer	Floating-Point
$\delta_{vtb}2^m$	2^m

Viterbi Computational Requirements		
Binary Compares	Binary Add./Sub.	Branch Metric Calculations
$2^m(2^k - 1)$	$2^m 2^k$	2^n

Table 3.2: Computational/Storage Requirements For a Typical Viterbi Decoder.

Additional Storage Required for Implementing a Post-Detector	
Integer	Floating-Point
$2\delta_{pd}2^m$	$2^m + \delta_{pd}2^k$

Additional Computation Required for Implementing a Post-Detector					
Operation	Binary Compares	Binary Add./Sub.	Brnch Met Calc's	Exp.	Mult./Div.
Viterbi$_{pd}$	$2^m(2^k - 1)$	$2^m 2^k$	2^n		
r.v. calc.		2^k			
Rev. Nec. ?	$\delta_{pd}(2^k - 1)$				
Comp. Past Dec.	$(\delta_{pd} \cdot 2^k)2^k$				
Select Min.	$(\delta_{pd} \cdot 2^k)2^k$				
Calc $P_r(sym)$		$2^k - 1$		2^k	2^k

Table 3.3: Additional Computation/Storage Required to Implement and Add a Generalized Post-Detector.

Computation requirements for the second Viterbi consist of performing the following for each received symbol: 2^n branch metric calculations, $2^m \cdot 2^k$ binary additions for path metric calculations, and $2^m(2^k - 1)$ binary comparisons. The calculation of the relative value vectors consist of 2^k subtractions per received symbol. To determine where revision is necessary, $\delta_{pd}(2^k - 1)$ comparisons are required. Assuming it is necessary for the entire path history length, this should impose at most $\delta_{pd} \cdot 2^k$ revisions each consisting of 2^k comparisons to examine the branch decisions made in the past, followed by at most 2^k binary comparisons for the "select min" operation. To obtain a normalized "roll-off" symbol probability that may later be used to deduce the input bit a posteriori probabilities will require 2^k exponentials, $2^k - 1$ additions, followed by 2^k divisions. These incremental computational costs are summarized in Table 3.3.

The overall computational requirements of implementing a SOVA utilizing the generalized post-detector algorithm can be determined by adding the various quantities presented in Table 3.2 and Table 3.3. For example, the number of binary additions required

	Generalized Post-Det.	MAP Algorithm	Best-Path Alg.
Storage:			
Integer	$(\delta_{vtb} + 2\delta_{pd})2^m$		
Float-Point	$2(2^m) + \delta_{pd}2^k$	$L_{msg}(2^m + 1)$	$L_{msg}(2^m + 1)$
Computation:			
Binary Compares	$2^{m+1}(2^k - 1) + \delta_{pd}(2^{2k+1} + 2^k)$		$2^{m+1}(2^k - 1)$
Binary Add./Sub.	$2^{m+1}2^k + 2^{k+1} - 1$	$2^{m+1}(2^k - 1)$	$2(2^{m+1}2^k)$
Brnch Met Calc's	2^{n+1}	2^n	2^n
Exponentials	2^k	2^n	2^m2^k
Mult./Div.	2^k	$2(2^{m+1}2^k)$	

Table 3.4: Comparison of the Computation Requirements of Various Soft-Output Decoder Algorithms.

is:

$$2^m2^k + 2^m2^k + 2^k + 2^k - 1 = 2^{m+1}2^k + 2^{k+1} - 1 \tag{61}$$

whereas the number of comparisons necessary has been approximated by:

$$\begin{aligned} & 2^m(2^k - 1) + 2^m(2^k - 1) + \delta_{pd}(2^k - 1) + (\delta_{pd} \cdot 2^k)2^k + (\delta_{pd} \cdot 2^k)2^k \\ \approx\ & 2^m(2^k - 1) + 2^m(2^k - 1) + \delta_{pd} \cdot 2^k + (\delta_{pd} \cdot 2^k)2^k + (\delta_{pd} \cdot 2^k)2^k \\ \approx\ & 2^{m+1}(2^k - 1) + \delta_{pd}(2^{2k+1} + 2^k) \end{aligned} \tag{62}$$

The number of branch calculations is 2^{n+1}. The number of exponentials and multiplication/division operations necessary each remain at 2^k. These results along with the computation and storage requirements of the MAP and Best-Path algorithms (see Appendix-C) are shown in Table 3.4.

By substituting the appropriate hardware and system parameters into Table 3.4, a system designer can determine which soft-output algorithm is most suited for his particular application. For example, suppose the available hardware can perform a binary comparison in one time unit, a binary addition in one time unit, and a binary multiplication in four time units. A time unit being defined as the duration of the fastest of those

three operations. Also, if a generalized post-detector is used, the decoders are to have a path memory of $\delta_{vtb} = \delta_{pd} = 5m$. (This should be sufficient since a code's constraint length is equal to at most m.) Finally, the transmitted sequence length in symbols is $L_{msg} = 256$. Then the computational and storage costs of the various algorithms can be compared by examining Figure 3.6 and Figure 3.7. In Figure 3.6, the amount of memory required to store the integer variables and the floating-point variables is assumed to be the same. Figure 3.6 depicts the amount of these "generic" storage elements required to implement the decoder versus code memory m and number of input bits k. Figure 3.7 depicts the amount of computation time required per decoded symbol vs. code memory m and number of input bits k. In Figure 3.7 the calculations of the branch metrics and exponentials were assumed to be performed by table lookup resulting in negligible computational costs.

A word of caution must be noted when interpreting the various tables and graphs. The actual amount of computation required depends not only on the number of additions, compares, multiplications, etc, but also on many other factors which are implementation specific. Just to name one, the tables and graphs did not take into account whether various operations were performed on integer or floating point quantities. The ratio of integers to floating-point variables required can be heavily influenced by the specific implementation and by constraints imposed upon the designer by the available hardware. Also note that the time required to move data from one storage location to another was completely neglected. The tables and graphs shown should only be used to provide a rough estimate of the amount of computation/storage required. If a designer can meet his computational/storage "budget" using worst case values for each operation (most likely floating point operations) then he can be fairly confident that the algorithm being considered can be implemented with the hardware at hand.

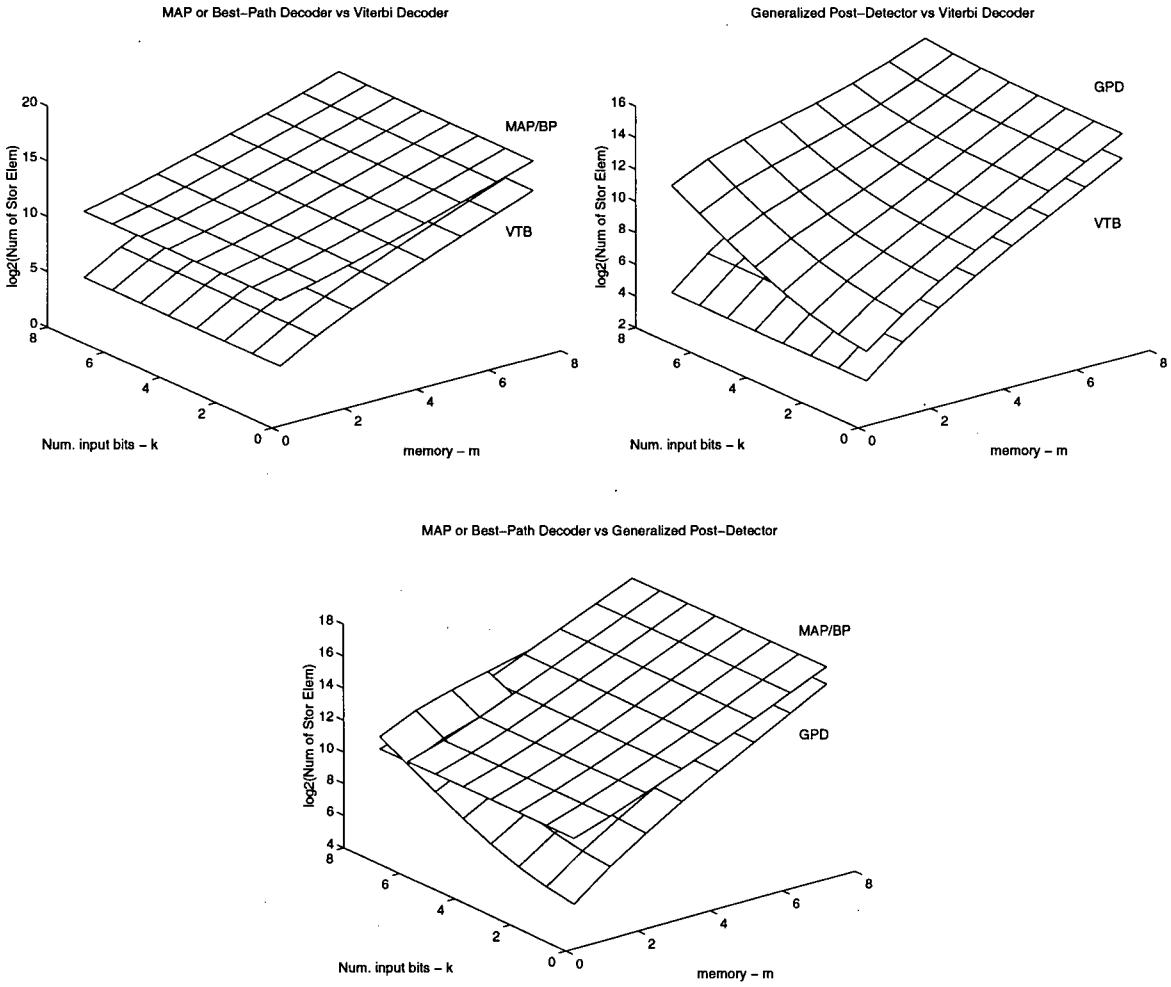


Figure 3.6: Comparisons of Algorithm Storage Requirements

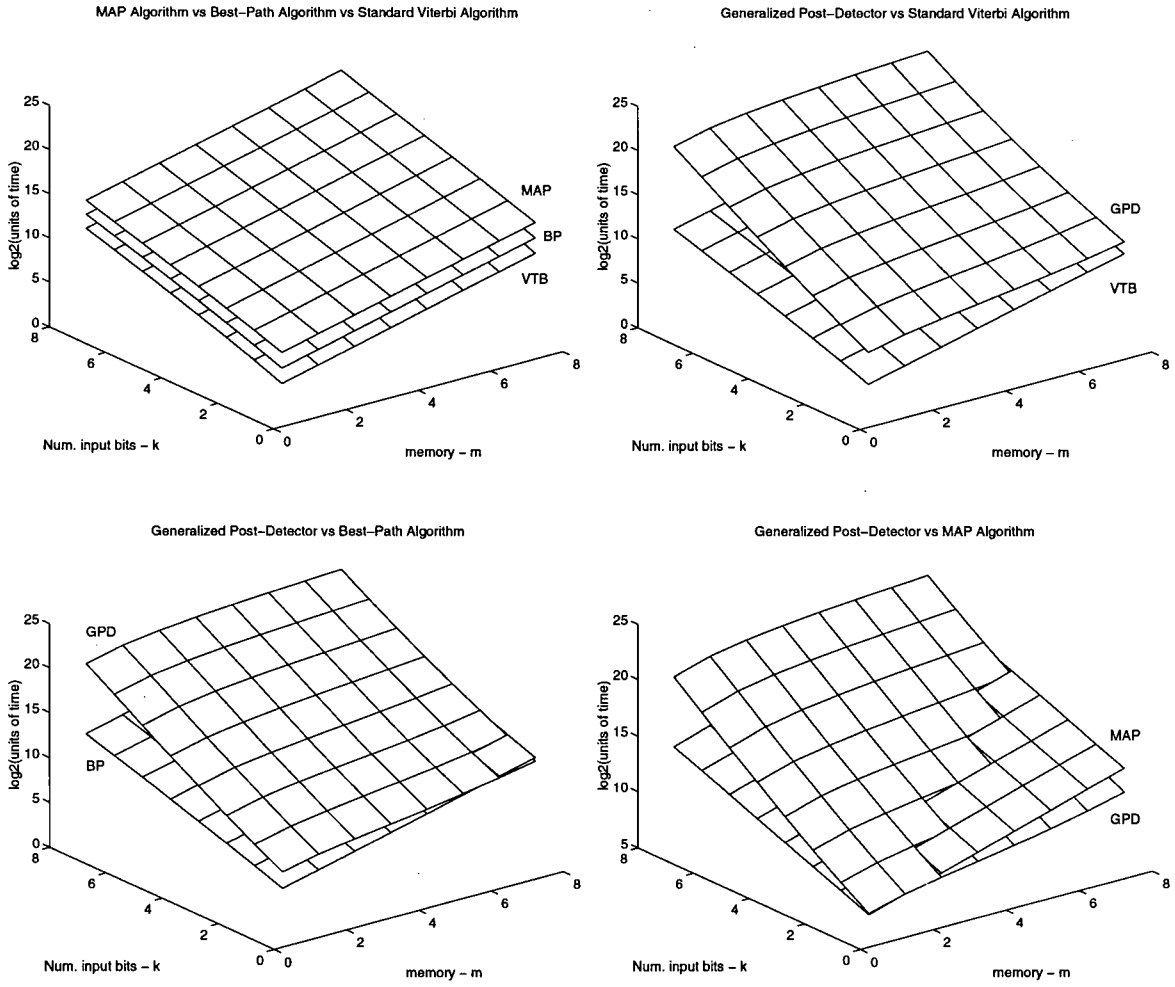


Figure 3.7: Comparisons of Algorithm Computational Requirements

Chapter 4

Evaluating Performance of The Generalized Post-Detector

4.1 Introduction

To determine how well the proposed post-detection algorithm would perform in actual practice, a computer model of a post-detector decoder was created. Simulations of various concatenated systems were performed covering a range of codes, signaling constellations, and channels. The simulations were arranged in sets such that each suite would vary a single system parameter such as the choice of inner code or choice of signaling constellation. In this way it would be possible to isolate the effects of each of these parameters on the post-detector's performance. The results of these simulations are described in the following sections.

In those sections, the results of the simulations are presented in the form of a graph depicting the resultant bit error rate (BER) vs SNR per input bit of the post-detection system. For comparison, the performance curve of the optimum serially concatenated system, as predicted by the graphical procedure described in [3] and Appendix-B, is also provided. In addition, the performance curve of a corresponding system not utilizing a soft-output inner decoder is also shown.

The codes and channels used for each of the simulations are referred to by a code or channel label such as cc2_3.df4 or isi2.70.30. cc2_3.df4 represents a a rate-$\frac{2}{3}$ convolutional code with a free distance $d_{free} = 4$ bits. Whereas isi2.70.30 represents an ISI channel with two taps: 70% of the received signal energy is transmitted via the first path while

30% of the energy is transmitted via a second delayed path. The delay is set to the symbol transmission interval. Unless otherwise stated, the channel tap weights are real (as opposed to complex value quantities). The generator matrices of each code and the tap weights of each ISI channel used for the simulations described in this thesis can be found in Appendix-C.

All of the computer models were created in C using the GNU C compiler. The code was compiled and run on either a Sun Sparcstation running SunOS 4.1.3, an AMD 486DX2-80 running FreeBSD 2.0.5, or a Cyrix 5x86-120 also running FreeBSD 2.0.5. In many cases, the same simulations were run on more that one computer system to verify the results. For all of the graphs shown, each point represents the transmission of at minimum 100000 bits and the counting of at least 400 bit errors in the received data stream. In order for a simulation to stop it had to meet both of these conditions. This should provide a sufficient number of error events to ensure statistical significance.

4.2 Effects of Code Free Distance on the Estimates of the A Posteriori Probabilities

To determine what effect code free distance has on the quality of the a posteriori probabilities generated by the post-detector, the simple concatenated coding scheme depicted in Figure 4.1 was implemented through software. Three convolutional codes with different free distances were selected for the inner code. To reduce the likelihood of selecting an outer code that was relatively insensitive to poor a posteriori probability estimates, three different convolutional codes were selected for the outer code. Every combination was implemented resulting in the nine graphs shown in Figures 4.2 - 4.4. Figure 4.2 shows the three cases where the inner code was set to cc2_3.df3. Figure 4.3 and 4.4 show the results of the simulations for the cases when the inner codes were fixed at cc2_3.df4

and cc2_3.df7 respectively.

For each simulation, a block type interleaver was used. The interleaver depth and width were each set to a value much greater than either of the codes constraint lengths and free distances. The modulation used for every case was BPSK. Transmission was through an AWGN channel. The revision history of the post-detectors and the path history memory of the Viterbi decoders were set to a value much greater than five times either of the code's constraint lengths ($> 20\times$).

Observe that all of the curves seem to show a similar result. As expected for a sub-optimal algorithm, the performance curve of the concatenated system using the proposed post-detector derived in Chapter-2 deviates from the ideal graphical prediction. This deviation can be characterized by a gradual "pulling away" from the ideal case as SNR increases. However, this deviation is quite small: less than 0.2 dB from the ideal case at bit error rates of 10^{-4}. This is quite acceptable when compared to the gain made over the system using a hard-output inner decoder.

The graphs indicate that there does not seem to be a simple relationship between the code free distance and the quality of the a posteriori information produced. The simulations do imply however that it should be possible to use the proposed post-detection algorithm on a variety of different convolutional codes regardless of the inner code's free distance.

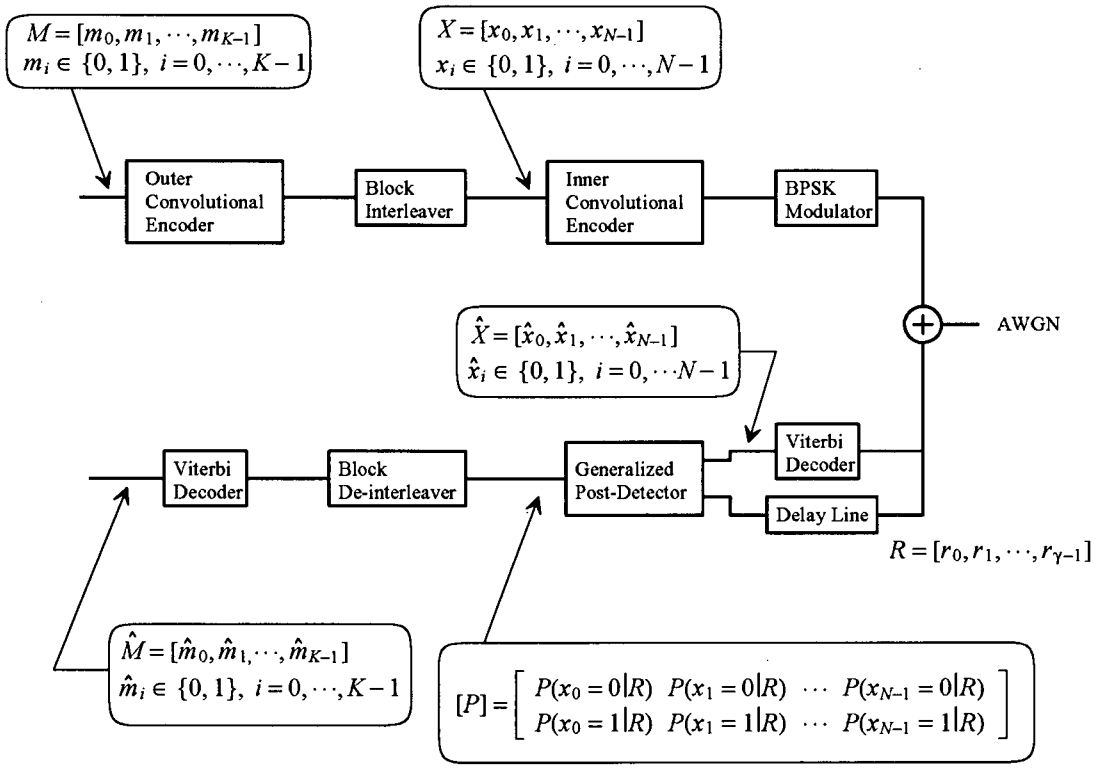


Figure 4.1: Concatenated Coding System Using Convolutional Codes for Both the Inner and Outer Codes.

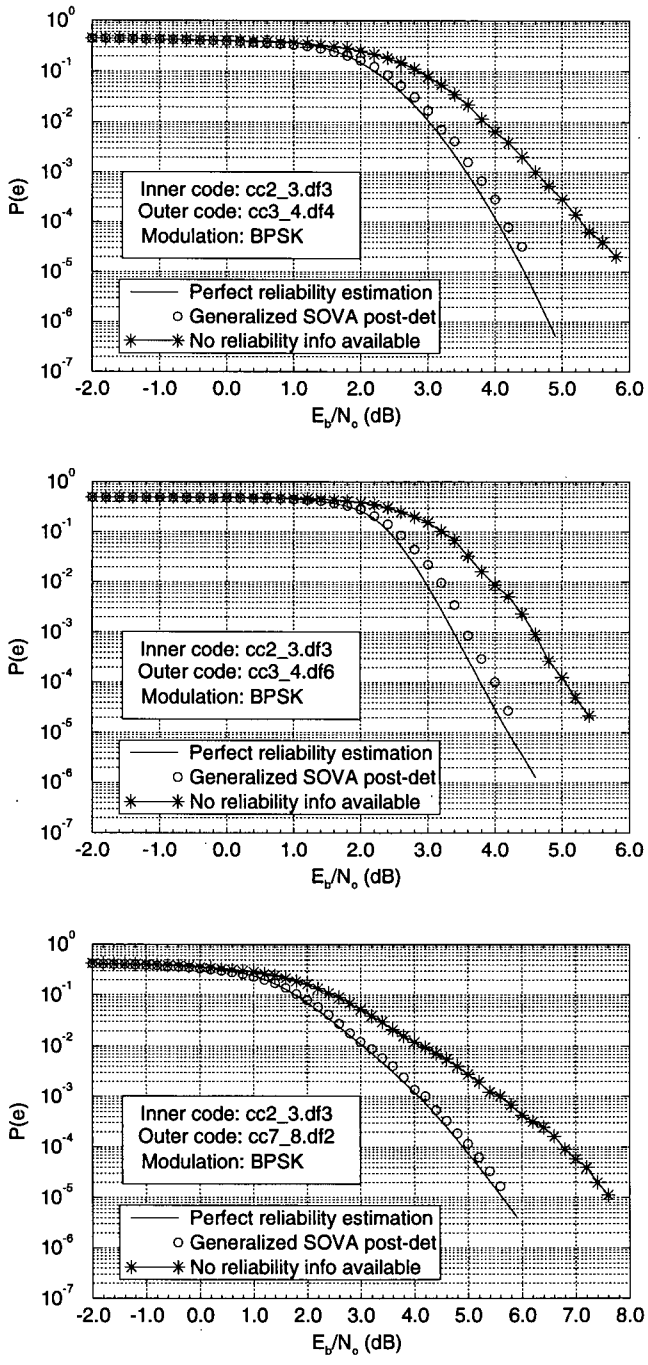


Figure 4.2: Performance of Various Concatenated Systems Utilizing an Inner Code with $d_{free} = 3$ (channel: AWGN).

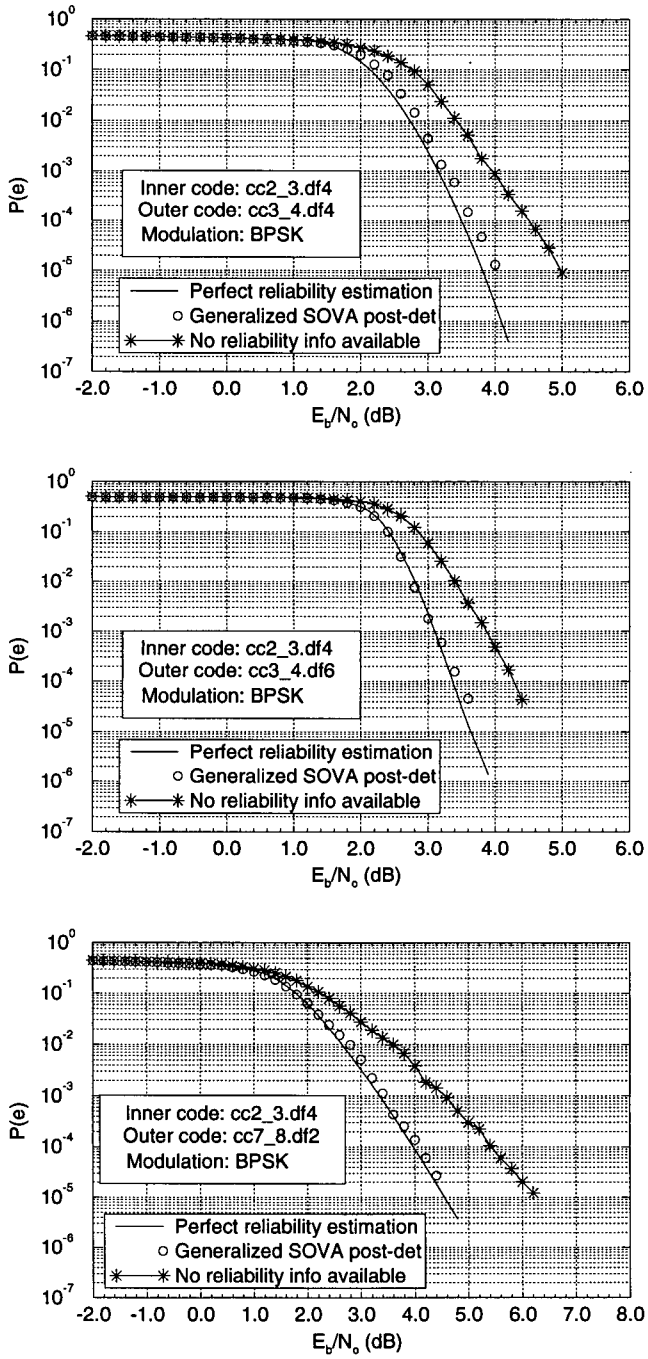


Figure 4.3: Performance of Various Concatenated Systems Utilizing an Inner Code with $d_{free} = 4$ (channel: AWGN).

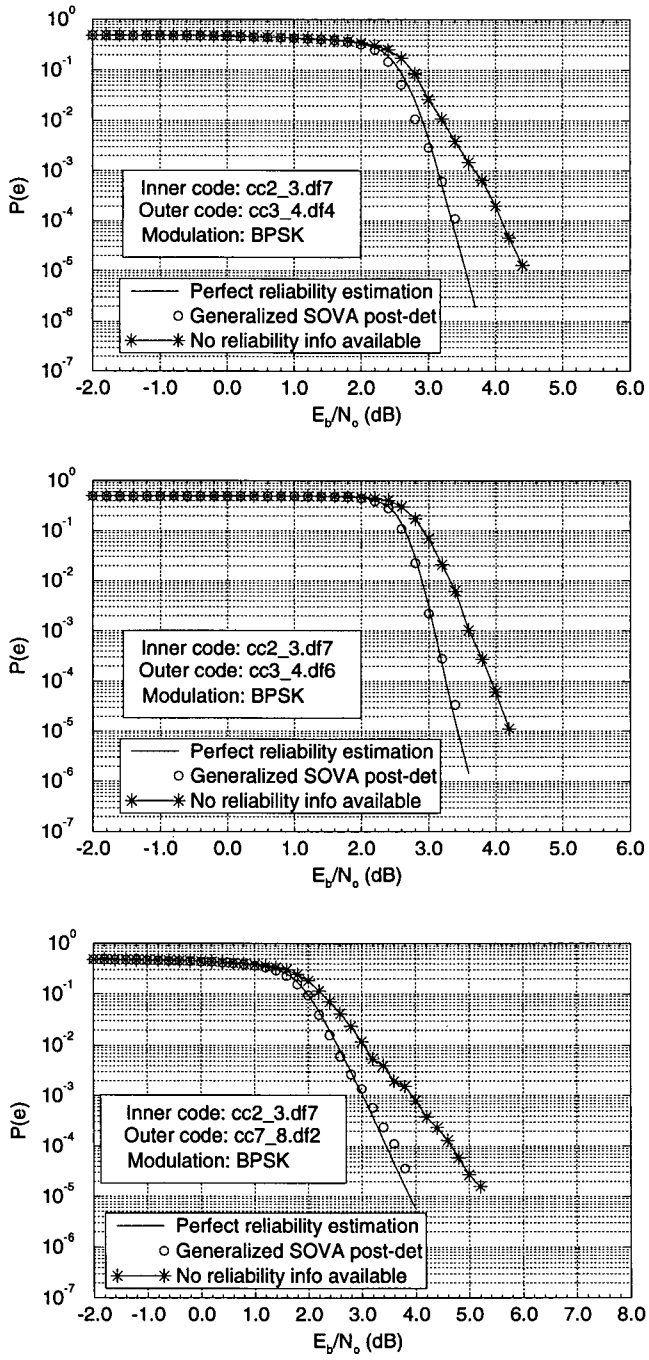


Figure 4.4: Performance of Various Concatenated Systems Utilizing an Inner Code with $d_{free} = 7$ (channel: AWGN).

4.3 Effects of Signaling Constellation on the Quality of the Estimates of the A posteriori Probabilities

To determine whether increasing the size of the signaling constellation would have any effect on the quality of the reliability information, software simulations of the concatenated CC/TCM system shown in Figure 4.5 were performed. Three simulations were performed. The first used coded 8-PSK modulation for the inner code, the second coded 16-QASK, while the third used coded 32-CROSS. Transmission was over an AWGN channel.

The results, shown in Figure 4.6 are quite similar to the results obtained in the previous section: a gradual increasing deviation with increasing SNR from the ideal graphically derived curve. As before, the deviation is small when compared to the gain made over the system utilizing the conventional Viterbi decoder for the inner code.

The graphs suggest that the proposed generalized post-detection algorithm should work reasonably well for a variety of different coded signaling constellations.

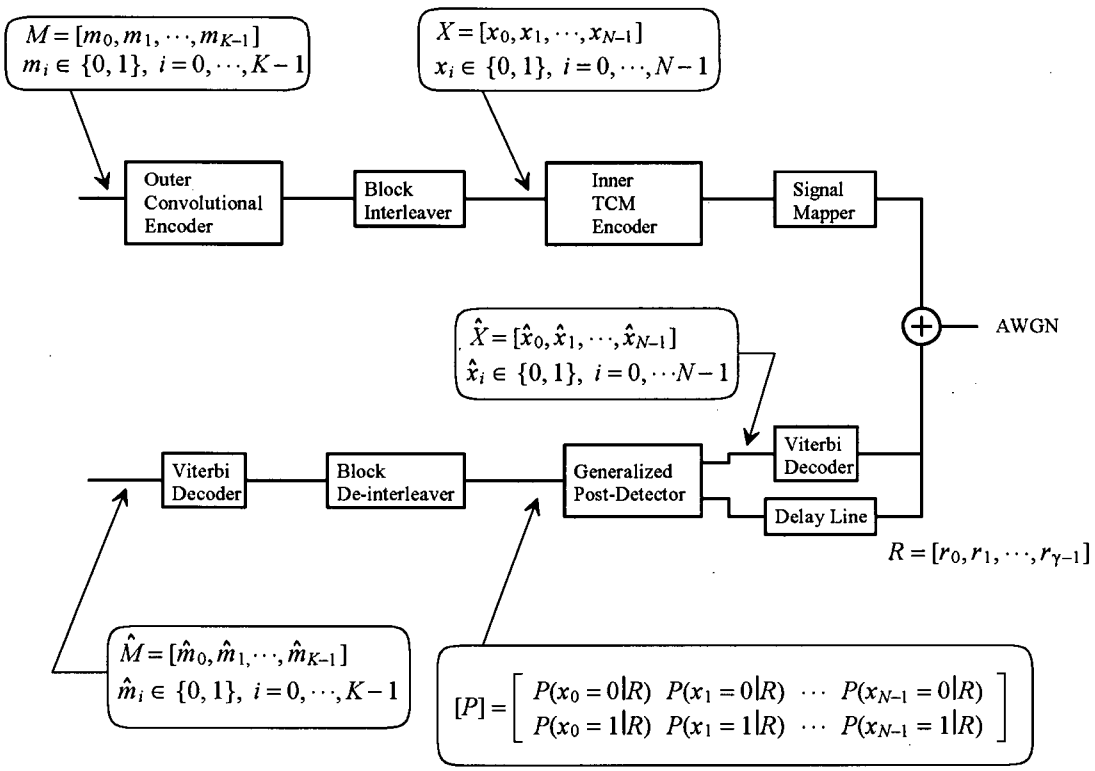


Figure 4.5: Concatenated Coding System using Multi-Level Modulation for the Inner Code.

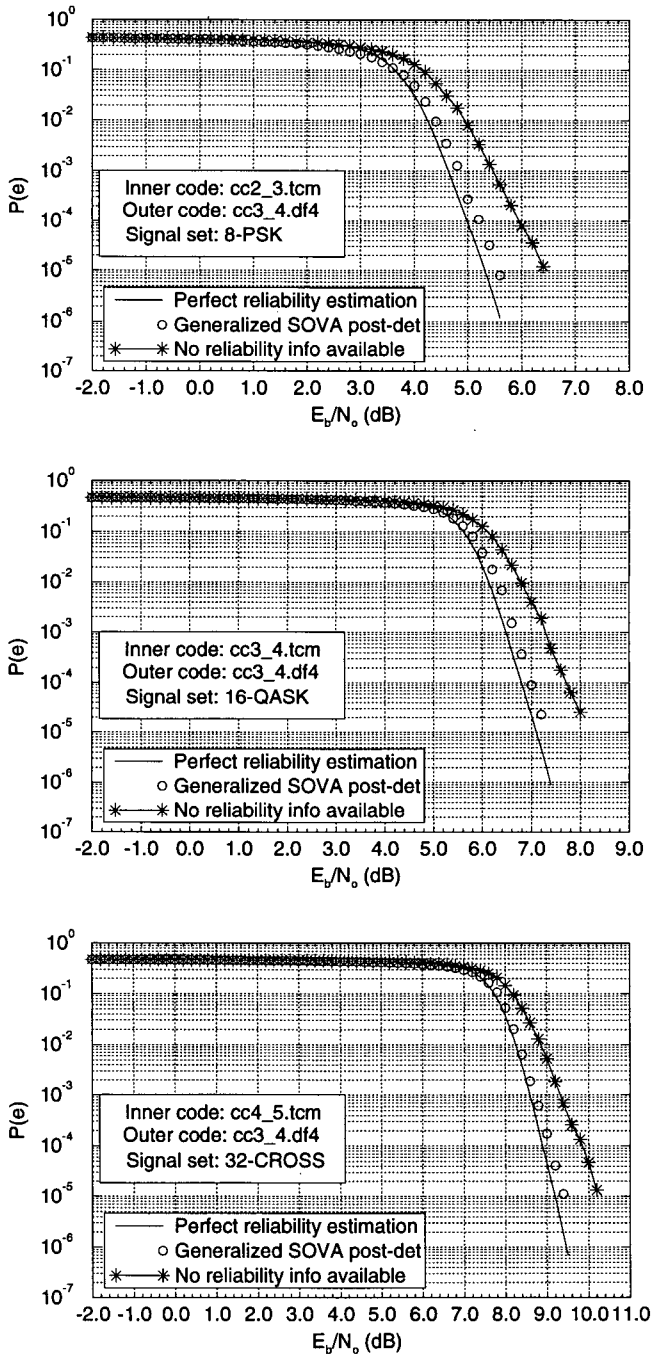


Figure 4.6: Effect of Different Signal Constellations on the Quality of Soft-Output Information (channel: AWGN).

4.4 Determining Performance of the Generalized Post-Detector for use in Soft-Output Viterbi Equalizers

To determine if the proposed post-detection algorithm can be used for implementing a soft-output Viterbi equalizer, the concatenated systems shown in Figure 4.7 and Figure 4.9 were tested via computer simulations. For these simulations, the soft-output Viterbi equalizers (SOVE) were given full knowledge of the channel tap weights. The channel tap weights were invariant with time. The interleaver depth and width were set to values much greater than the constraint length of either the "outer" code or the ISI channel. The post detector revision length and Viterbi decoder path memories were all much greater than five times the constraint length of either the code or the channel.

The first suite of simulations was used to determine the effects of the different ISI channel tap weights on the quality of the a posteriori information produced by the soft-output Viterbi equalizer. For these simulations, binary data was encoded using the convolutional code cc2_3.df4 and transmitted over the various ISI channels using 8-PSK modulation. The deduced a posteriori 8-PSK symbol probabilities were used to calculate a posteriori input bit probabilities of the modulator. These bit probabilities were in turn used for the decoding of the convolutional code. Hence, the convolutional code was decoded as if binary modulation had been used – even though, the probabilities themselves were originally deduced using the received 8-PSK symbols. The ISI channel was comprised of two taps only. The delay between the taps was one 8-PSK symbol interval.

The results of these simulations are depicted in Figure 4.8. Observe that the quality of the soft-output information appears to degrade with increasing ISI. However, even for the worst case scenario (two taps of equal weight), the degradation is not too severe. Therefore the simulations indicate that for moderate levels of ISI the quality of the a

posteriori probability estimates is quite good.

For the second suite of simulations a three tap ISI channel is used for the inner code. In addition, the outer encoder is replaced by a TCM modulator (coded 8-PSK). As a result, the outer decoder would need to have knowledge of the a posteriori input symbol probabilities (as opposed to the bit probabilities used for the first suite of simulations).

Results from the second suite of simulations are shown in Figure 4.10. For comparison, the results of equivalent systems utilizing the MAP and Best-Path algorithms are also provided in addition to the graphically predicted curve. The most notable characteristic of these simulations is that for the system with a higher level of ISI (inner code: isi3.60.20.20), the resultant curve of the MAP based system does not coincide with the curve generated by the graphical procedure described in [3]. Yet for the system with less ISI (inner code: isi3.80.10.10) the MAP based system does appear to agree with the graphically predicted curve. One possible explanation is as follows. The graphical procedure described in [3] assumes that the inner code can be modeled by an equivalent Gaussian channel. Referring to equation (6) and (10) in section 2.1, the model assumes that:

$$\prod_{j=1}^{L} P(q_{ji}(\bar{m}_i)|\bar{R}_j) = \mathcal{C} \prod_{j=1}^{L} f(Y_{ij}|p_{ij}(\bar{m}_i)) \tag{63}$$

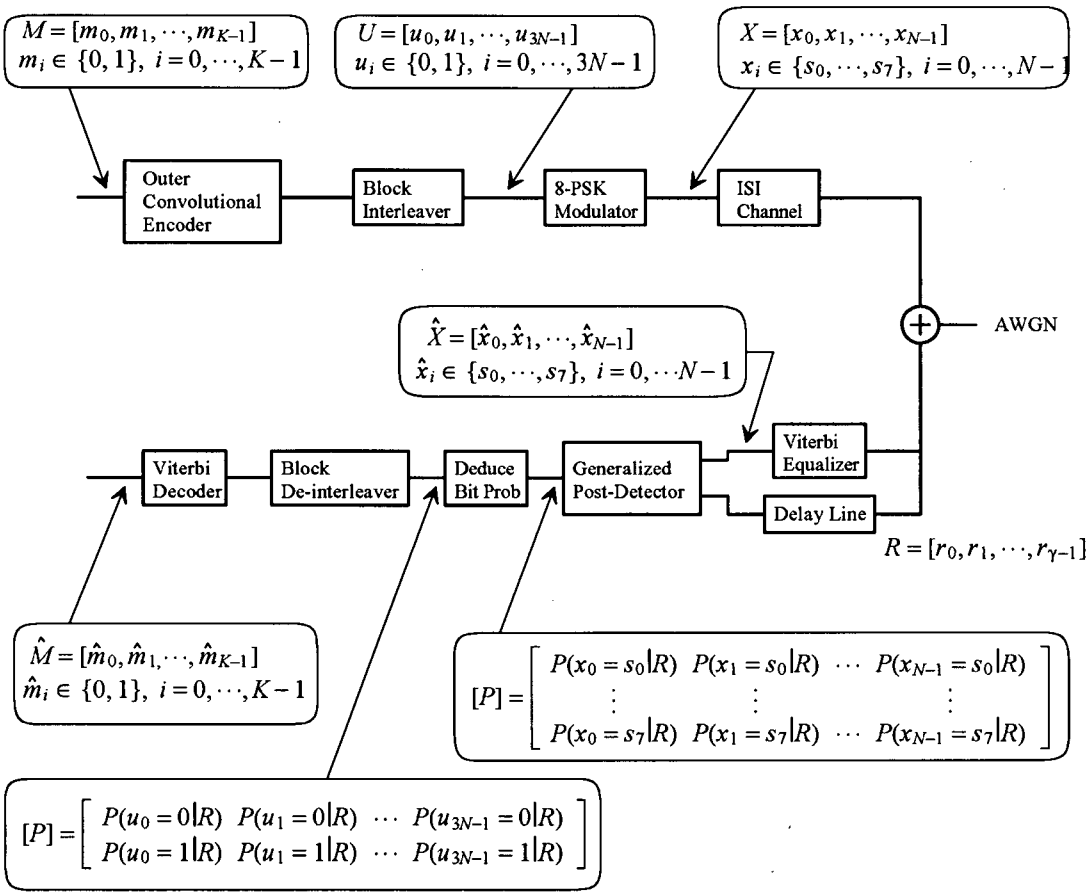
$$= \mathcal{C} \prod_{j=1}^{L} \frac{1}{\sqrt{2\pi\nu^2}} \exp\left(\frac{|Y_{ij} - p_{ij}(\bar{m}_i)|^2}{2\nu^2}\right) \tag{64}$$

where $\mathcal{C}$ is some constant and ν is an equivalent noise variance. The fact that for the high ISI case, the MAP based curve does not coincide with the graphically deduced curve would suggest that it is not possible to find an equivalent sequence $\bar{Y}_i$ that can satisfy equation (64). When the outer code utilizes non-binary signaling it would seem that there are not enough degrees of freedom to find a suitable fit. Why then, does the MAP based curve match the graphically deduced curve for the low ISI case? This may be explained

by the following reasoning. The one situation where equation (64) should hold is for a channel with no ISI. In that particular case, the equalizer should not do anything and should not affect the channel statistics. Therefore, a MAP based simulation with such a channel (inner code: isi3.100.0.0) should match perfectly with the graphically produced curve. However, as ISI increases, the statistics produced by the soft-output decoder seem to become less Gaussian and result in a deviation of the MAP based simulation curve from the graphically produced curve. This deviation would increase as the level of ISI over the inner channel increases. This explanation seems to be consistent with the observed results from the simulations.

Overlooked in the previous discussion is the fact that the generalized post-detection algorithm seems to also work reasonably well for SOVE in systems that utilize non-binary modulation. The deviation from the optimum curve (determined by the MAP based system for this case) is not too unreasonable. It displays the same characteristic behaviour as was seen in the first suite of simulations. The deviation from the ideal case increases with increasing ISI.

The results of the previous two suites of simulations suggest that for moderate levels of ISI, the quality of a posteriori probabilities produced by the proposed generalized post-detection algorithm is quite good. This is a somewhat surprising result considering the concerns that were raised in Section 3.6.2 . One can only conclude that the approximations made during the derivation of the generalized post-detection algorithm are quite reasonable when the transmitted sequence from the "inner" code (in this case, it is an ISI channel) is transmitted over an AWGN channel.



Notes: (1) ISI channel taps are static.
(2) ISI channel tap weights are known by the equalizers.

Figure 4.7: Concatenated Coding System with Uncoded 8-PSK Modulation Over a Static ISI Channel.

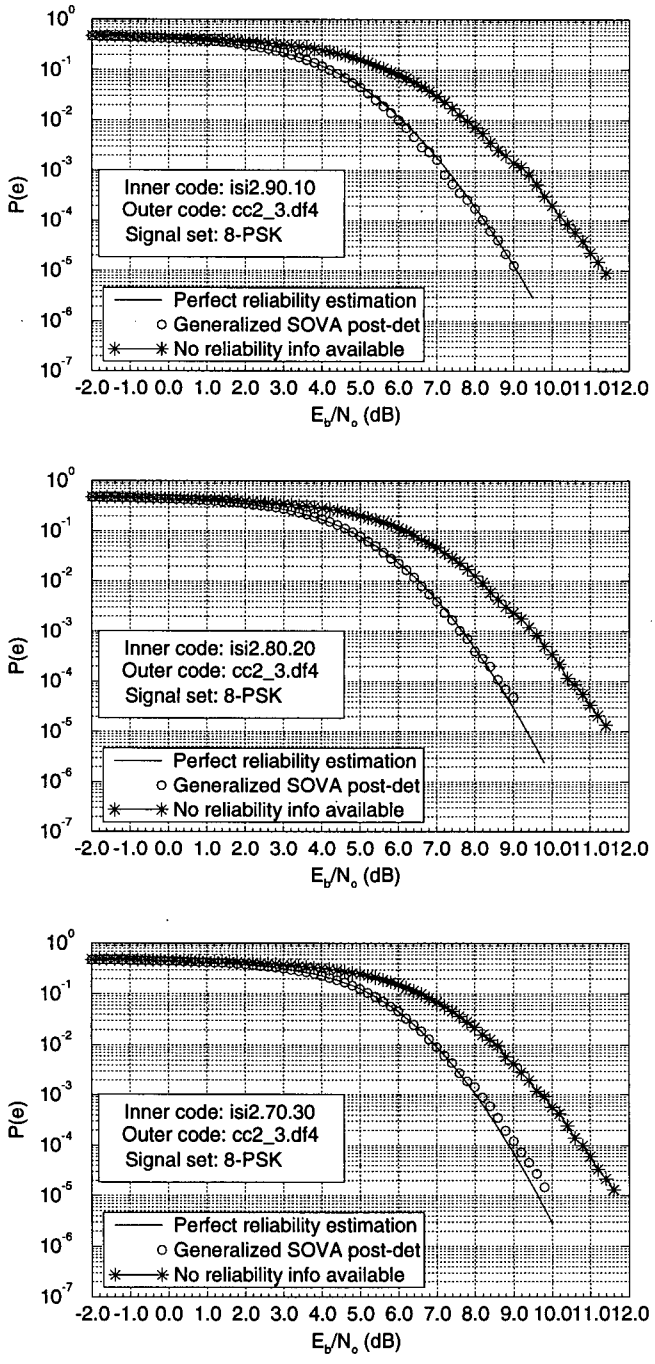


Figure 4.8: Soft-Output Equalizer Performance in a Concatenated System Using Uncoded 8-PSK for the Outer Code.

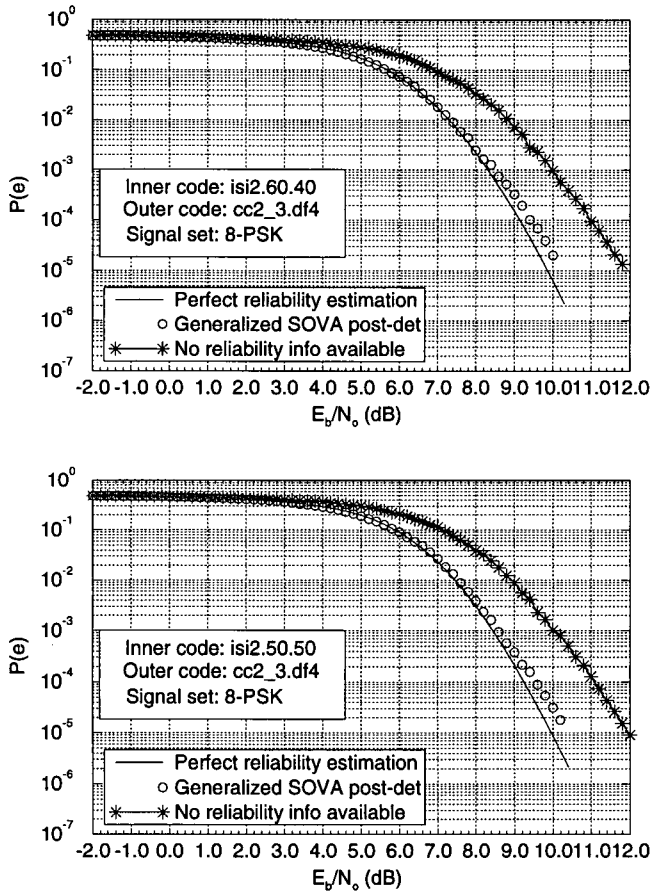
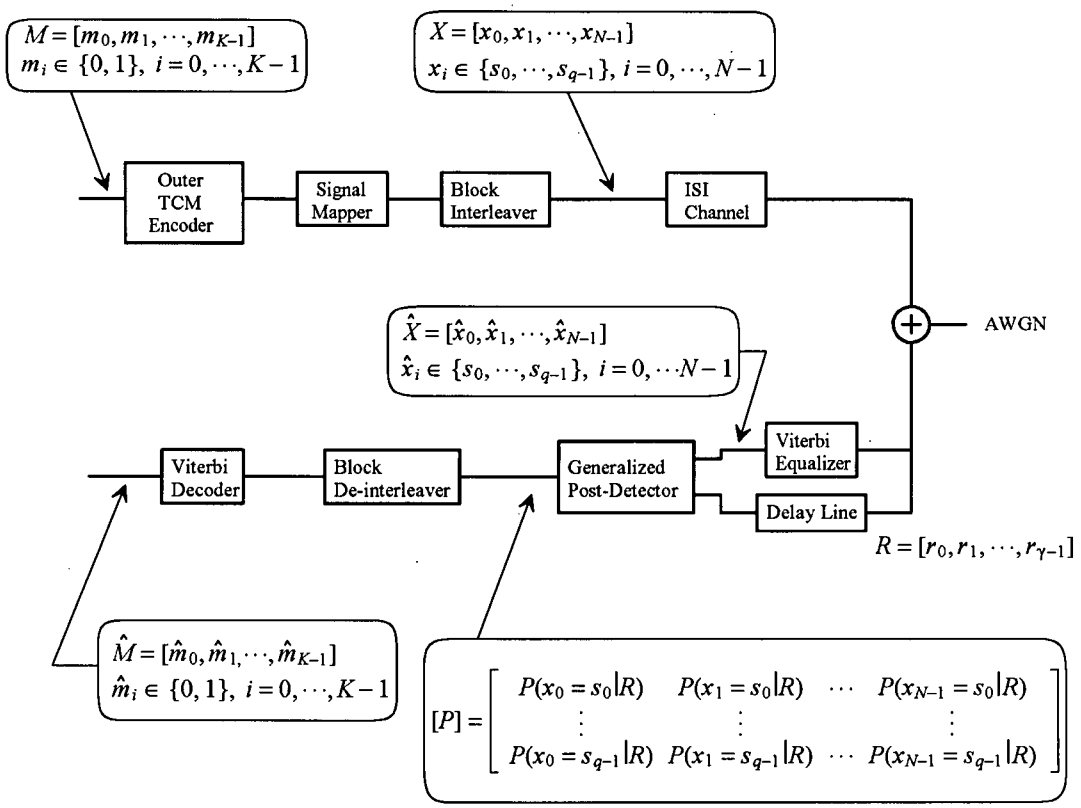


Figure 4.8: *Continued.*



Notes: (1) ISI channel taps are static.
(2) ISI channel tap weights are known by the equalizers.

Figure 4.9: Concatenated System Using Coded 8-PSK Over a Static ISI Channel.

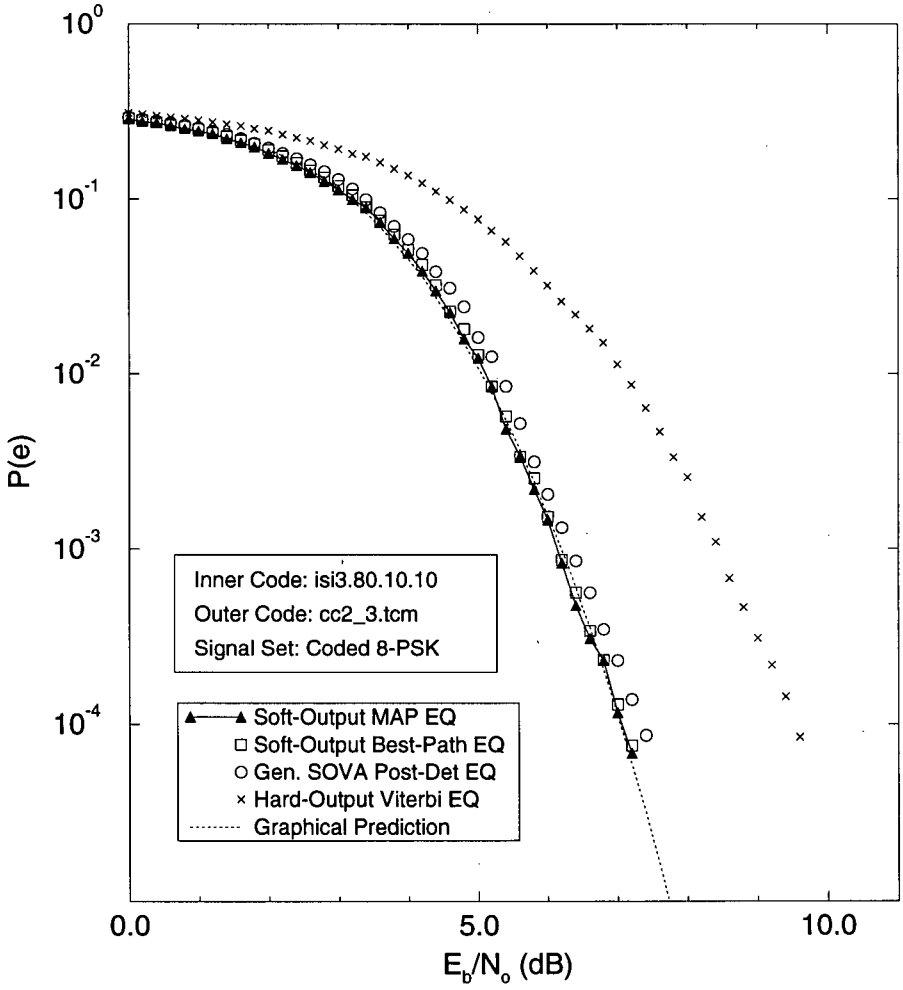


Figure 4.10: Soft-Output Equalizer Performance in a Concatenated System Using Coded 8-PSK for the Outer Code.

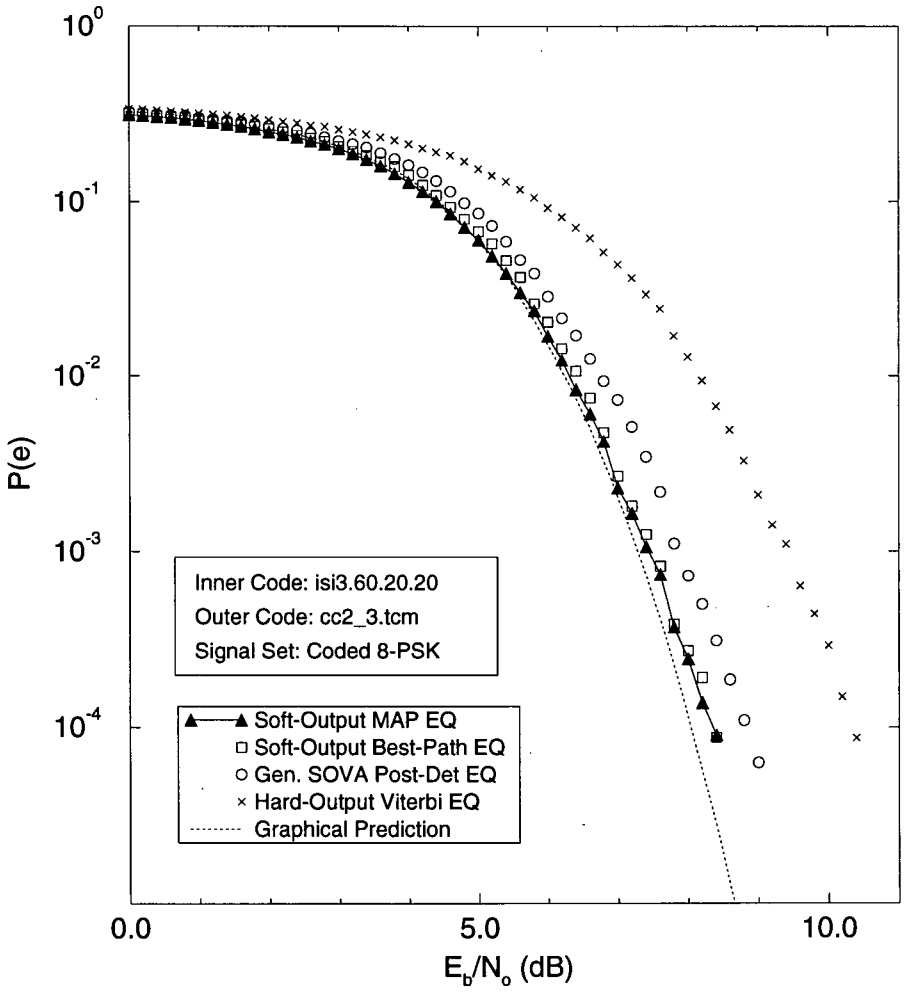


Figure 4.10: *Continued.*

4.5 Determination of the Minimum Required Revision History for the Generalized Post-Detector

To determine what is the minimum revision depth for the generalized post-detector in order for it to provide reasonably good estimates of the a posteriori input probabilities, the concatenated system shown in Figure 4.1 was once again simulated via computer software. However, this time the revision depth of the post-detector was varied between $1\times$ and $10\times$ the inner code's constraint length. The length of the Viterbi path history arrays were kept at a value much greater than $5\times$ the appropriate code's constraint length.

The results of these simulations are shown in Figure 4.11. The curves show the effect of changing the revision depth (and hence required memory) of the generalized post-detector. The curves indicate that a generalized post-detector should generate reasonable good quality a posteriori input probability estimates if its revision depth is set to a value of $\delta_{pd} \geq 5 \times$ (constraint length).

This result is not unexpected since the proposed algorithm does not revise the relative value vector for a given memory depth index if all of the possible paths yielded the same decision at that index. Since all of the surviving paths of a Viterbi decoder tend to merge by $5\times$ the code's constraint length, clearly revision beyond this point is generally not done.

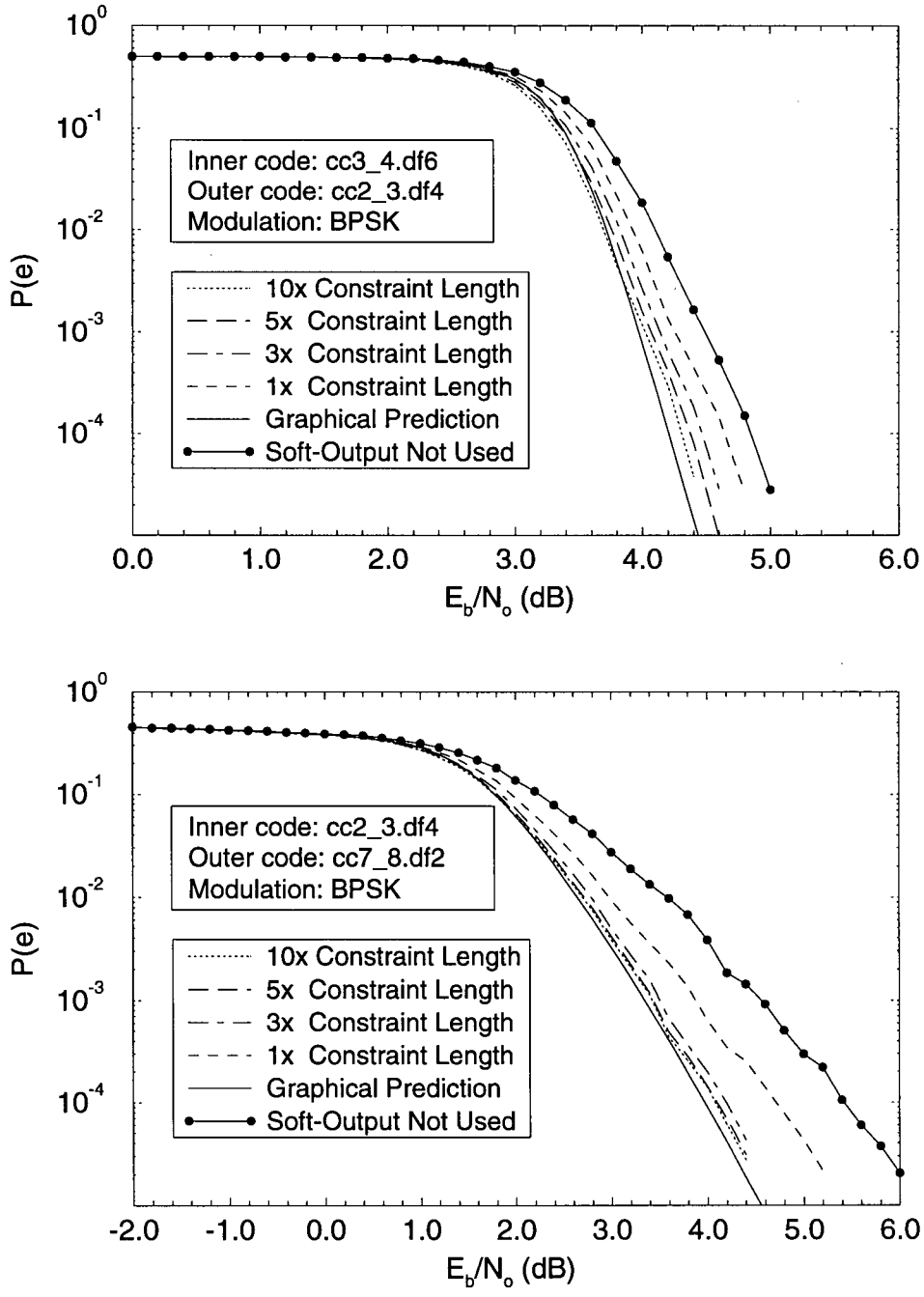


Figure 4.11: Simulation Results of a Concatenated System Utilizing a Post-Detector With Various Revision Depths (channel: AWGN).

Chapter 5

Conclusion

5.1 Summary

This thesis tackles the problem of finding an efficient heuristic algorithm for deducing estimates of the a posteriori input probabilities of a Markovian based encoder. When this project was begun, the better known variants of this approach were applicable to $(n, 1, m)$ codes only. Therefore there existed a need to find an efficient heuristic based algorithm that would be applicable to the more general case of (n, k, m) codes.

Of the known $(n, 1, m)$ heuristically based soft-output decoders, the most efficient of these in regards to computational and memory storage requirements is the decoder proposed by Berrou *et al.* [3]. The key to this decoder's efficiency was that it utilized a post-detector architecture. Berrou *et al.* were able to find a simple $(n, 1, m)$ revision algorithm that was compatible with this type of decoder architecture. The question naturally arose of whether it would be possible to find a (n, k, m) revision formula that was also compatible with this particular architecture. Fortunately, it was indeed possible.

Finding such an algorithm involved several steps. The first was to use several approximations to simplify Battail's general heuristic revision formula [2] such that its complexity would be reduced to a reasonable level. The next step involved a comparison of the revision operations dictated by this simplified formula (for the $(n, 1, m)$ case) with the revision operations dictated by the $(n, 1, m)$ post-detector algorithm published by Berrou *et al.* These comparisons resulted in the determination of what additional

modifications had to be made to the simplified Battail revision formula in order for it to be able to take advantage of the post-detector decoder architecture. The resulting algorithm is the "generalized post-detector algorithm" described in Chapter-3.

Simulations show that the proposed algorithm is capable of providing reasonably high quality estimates of a posteriori input probabilities for a variety of different convolutional codes. Furthermore, despite some concerns raised in Section 3.6.2, the proposed algorithm also seems adequate for use with Viterbi equalizers. In addition, it was found that a revision depth of $5 \times$ (constraint length) should be sufficient for many applications. For the serial concatenation of two codes, performance can be characterized as follows. Approximately 80% of the maximum possible SNR (dB) gain attainable through the use of soft-decision decoding for each stage is achieved (the best performance being arrived at through the use of a soft-output MAP inner decoder and the worst through the use of a standard hard-output inner decoder). This seems to be typical of heuristic based algorithms.

Benefits of using the new algorithm include the fact that it can be applied to any (n, k, m) trellis based code. The decoding delay is independent of the length of the transmitted code sequence and can be as short as $10 \times$ (constraint length). Disadvantages of this algorithm include the fact that the quality of a posteriori input probability estimates is not as good as those produced by the MAP or Best-Path algorithms. Since the algorithm is based on the Viterbi algorithm it shares the same limitations of the Viterbi algorithm: practical only for relatively short constraint length codes. Finally, knowledge of the channel variance is required by the receiver to calculate the branch metrics (a problem shared by the MAP and Best-Path algorithms as well).

5.2 Proposed Future Work

One of the main incentives for the renewed interest in soft-output decoders is that they are an essential element of Turbo decoders[4]. It would be worthwhile to determine how well the new algorithm performs for this particular application. One question that arises is whether it is even worthwhile to consider a sub-optimal soft-output algorithm. It may be that for a given level or performance, the turbo decoder may have to perform more iterations with a SOVA type algorithm than it would with a MAP type algorithm. This might offset any computational advantages that the SOVA algorithm may have held.

For this thesis, the main objective of the computer simulations was to determine how certain system parameters would affect the quality of the a posteriori input probabilities deduced by the generalized SOVA. Several concatenated systems were simulated in which individual system parameters such as d_{free} or the choice of signaling constellation could be isolated and changed. Consequently, many of the simulated systems seem artificial in that one would never choose those combinations of "codes" for actual applications. It would be desirable to perform some simulations that are more indicative of "real world" systems. The codes selected should be representative of codes used in actual practice. On a related note, it would be desirable to determine how real world channel impairments affect the quality of the soft-output information. For example, simulations should be performed over the Rayleigh and Rician fading channels.

Finally, incidental to the main goal of the work described in this thesis, it was discovered that the graphical algorithm described by Berrou *et al.* [3] may not accurately predict the performance of a concatenated system that must directly use the a posteriori input symbol probabilities for calculating the metrics of the outer decoder. It is hypothesized that this is due to the deduced a posteriori input symbol probabilities having non-Gaussian statistics for these cases. This should be verified.

References

[1] L. R. Bahl, J. Cocke, F. Jelinek, and J. Raviv, "Optimal Decoding of Linear Codes for Minimizing Symbol Error Rate," *IEEE Trans. Inform. Theory,* Vol. IT-20, No. 2, pp. 284-287, March 1974.

[2] G. Battail, "Pondération des symboles décodés par l'algorithme de Viterbi," (in French), *Annales des Télécommunications,* Fr., 42, N° 1-2, pp. 31-38, January 1987.

[3] C. Berrou, P. Adde, E. Angui, and S. Faudeil, "A Low Complexity Soft-Output Viterbi Decoder Architecture," *1993 IEEE Internat. Conf. on Comm.*, Vol. 2, pp. 737-740.

[4] C. Berrou, A. Glavieux, P. Thitimajshima, "Near Shannon Limit Error Correcting Coding and Decoding: Turbo-codes," *IEEE Int. Conference on Comm, ICC'93* Geneva, Switzerland, Vol 2/3, pp. 1064-1070, May 1993.

[5] D. G. Daut, J. W. Modestino, L. D. Wismer, "New Short Constraint Length Convolutional Code Constructions for Selected Rational Rates," *IEEE Trans. Inform. Theory,* Vol. IT-28, No. 5, pp. 794-800, September 1982.

[6] J. Hagenauer, P. Hoeher, "A Viterbi Algorithm with Soft-Decision Outputs and its Applications," *Proc. of IEEE Globecom'89,* Dallas, Texas, pp. 47.1.1-47.1.7, November 1989.

[7] P. A. Humblet, "Efficient Maximum-a-Posteriori Symbol Detection," submitted for publication, Institut Eurecom, Sophia-Antipolis, France, July 1994.

[8] G. D. Forney, JR., "Maximum-Likelihood Sequence Estimation of Digital Sequences in the Presence of Intersymbol Interference," *IEEE Trans. on Inform. Theory,* Vol. IT-18, No. 3, pp. 363-378, May 1972.

[9] G. D. Forney, JR., "The Viterbi Algorithm," *Proc. of the IEEE* , Vol. 61, No. 3, pp. 268-278, March 1973.

[10] Y. Jain, "Convolutional Codes Improve Bit-Error Rate in Digital Systems," *EDN* pp. 129-133, August 20, 1990.

[11] S. Lin, D. J. Constello Jr., "Error Control Coding: Fundamentals and Applications," Prentice-Hall, Inc., Englewood Cliffs, N.J., 1983.

[12] J. G. Proakis, "Digital Communications, Second Edition," McGraw-Hill, Inc, New York, N.Y., 1989.

[13] G. Ungerboeck, "Trellis-Coded Modulation with Redundant Signal Sets," *IEEE Communications Magazine,* Vol. 25, No. 2, pp. 5-21, February 1987.

[14] X. Wang, S. B. Wicker, "A Soft-Output Decoding Algorithm for Concatenated Systems," *IEEE Trans. on Inform. Theory,* Vol. IT-42, No. 2, pp. 543-553, March 1996.

Appendix A

Predicting Concatenated Code Performance - Graphical Method

Consider the concatenated system depicted in Figure A.1. The output of the inner soft-output decoder is a discrete-time continuous random variable. If sufficient interleaving is used there should be little correlation amongst the errors entering the outer decoder. If the assumption is made that any decoding metric based upon these samples is Gaussian distributed then it is possible to treat the inner-code/interleaver combination as an equivalent discrete-time AWGN channel. This is the basis of the graphical procedure described in [3] used to predict the bit-error rate (BER) performance of a concatenated system.

To use this procedure it is necessary to determine the relationship between the signal-to-noise ratio (SNR) of the global concatenated system and the SNR of a system comprised of the inner code only. If E_b^{global} and E_b^{inner} represent the transmitted energy per bit of the global concatenated system and inner coding system respectively, while E_s represents the energy per symbol transmitted over the channel then:

$$E_b^{global} = \frac{E_s}{R_g}, \quad \text{where } R_g = R_o R_i \tag{65}$$

$$E_b^{inner} = \frac{E_s}{R_i} \tag{66}$$

Therefore,

$$\frac{E_b^{inner}}{N_o} = \frac{E_b^{global}}{N_o}\frac{R_g}{R_i} \tag{67}$$

Now letting $SNR = 10 \log \frac{E_b}{N_o}$ it follows that:

$$SNR^{inner} = SNR^{global} - \Delta, \quad \text{where } \Delta = 10 \log \frac{R_i}{R_g} = 10 \log \frac{1}{R_o} \tag{68}$$

To determine the BER of the concatenated system at a specified global SNR (point-a in figures A.1 and A.2) perform the following procedure. Use equation (68) to determine the SNR of the inner code system (point-b). Look-up in a graph or table the resultant BER of the inner system (point-c). Determine what SNR would be required to achieve this BER on an equivalent uncoded AWGN channel (point-d). (In effect, the inner coding system is being emulated by an equivalent AWGN channel.) Once again, using equation (68), translate this SNR back to the equivalent system's overall SNR (point-e). Using a graph or table of the BER performance of the outer system operating over a standard AWGN channel use the equivalent system's overall SNR to look-up the resultant BER of the outer code (point-f). The abscissa (x-coordinate) of point-a and the ordinate (y-coordinate) of point-f specify an operating point of the concatenated system (point-g).

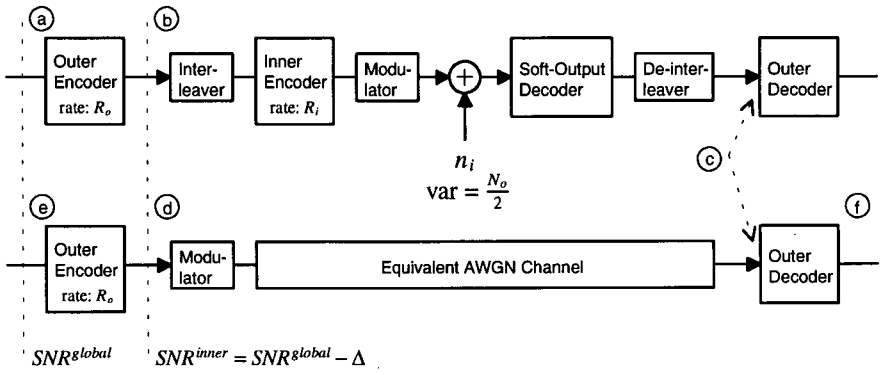


Figure A.1: System Model for Graphical Procedure

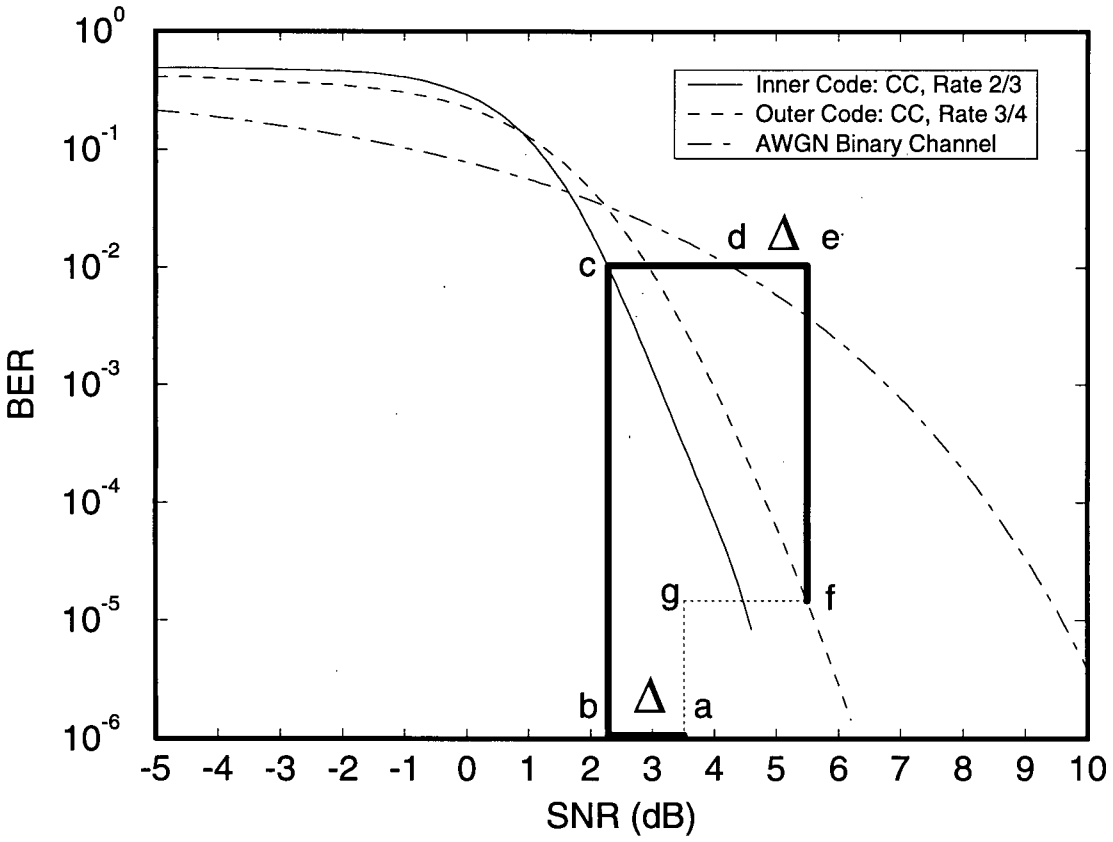


Figure A.2: Graphical Determination of Concatenated System Performance

Appendix B

Code Tables

The following tables describe the codes associated with each "code label" referenced in the simulations shown in Chapter-4.

B.1 Convolutional Codes

Label	Rate	C.L. (symbols)	Tot Mem (bits)	Generator Sequences (octal)	d_{free}	Reference
cc2_3.df3	$\frac{2}{3}$	1	2	6 2 6 2 4 4	3	[11]
cc2_3.df4	$\frac{2}{3}$	2	3	4 2 6 1 4 7	4	[11]
cc2_3.df7	$\frac{2}{3}$	3	6	64 30 64 30 64 74	7	[11]
cc3_4.df4	$\frac{3}{4}$	2	3	4 4 4 4 0 6 2 4 0 2 5 5	4	[11]
cc3_4.df6	$\frac{3}{4}$	2	6	6 1 0 7 3 4 1 6 2 3 7 4	6	[11]
cc7_8.df2	$\frac{7}{8}$	1	1	2 0 2 0 2 0 6 6 0 0 0 0 0 4 0 4 0 0 0 0 4 0 0 4 0 0 0 4 0 0 0 4 0 0 4 0 0 0 0 4 0 4 0 0 0 0 0 4 4 0 0 0 0 0 0 4	2	[Daut et al]

B.2 TCM Codes

Label	C.L. (sym)	Tot Mem (bits)	Generator Sequences (octal)
cc2_3.tcm	2	2	4 0 0 0 7 5
Mapping:	$0 : +\cos(22.5°) - j\sin(22.5°)$ $1 : -\sin(22.5°) + j\cos(22.5°)$ $2 : +\cos(22.5°) + j\sin(22.5°)$ $3 : +\sin(22.5°) + j\cos(22.5°)$ $4 : -\cos(22.5°) + j\sin(22.5°)$ $5 : +\sin(22.5°) - j\cos(22.5°)$ $6 : -\cos(22.5°) - j\sin(22.5°)$ $7 : -\sin(22.5°) - j\cos(22.5°)$		
Notes:	Feed-forward version of code presented in [13]		

Label	C.L. (sym)	Tot Mem (bits)	Generator Sequences (octal)
cc3_4.tcm	2	3	4 0 0 0 0 4 2 0 0 3 5 2
Mapping:			$0: +\frac{3}{\sqrt{10}} - j\frac{1}{\sqrt{10}}$ $1: +\frac{1}{\sqrt{10}} + j\frac{3}{\sqrt{10}}$ $2: +\frac{1}{\sqrt{10}} - j\frac{3}{\sqrt{10}}$ $3: +\frac{3}{\sqrt{10}} + j\frac{1}{\sqrt{10}}$ $4: +\frac{3}{\sqrt{10}} + j\frac{3}{\sqrt{10}}$ $5: -\frac{3}{\sqrt{10}} + j\frac{3}{\sqrt{10}}$ $6: +\frac{1}{\sqrt{10}} + j\frac{1}{\sqrt{10}}$ $7: +\frac{3}{\sqrt{10}} - j\frac{3}{\sqrt{10}}$ $8: -\frac{1}{\sqrt{10}} + j\frac{3}{\sqrt{10}}$ $9: -\frac{3}{\sqrt{10}} - j\frac{1}{\sqrt{10}}$ $10: -\frac{3}{\sqrt{10}} + j\frac{1}{\sqrt{10}}$ $11: -\frac{1}{\sqrt{10}} - j\frac{3}{\sqrt{10}}$ $12: -\frac{1}{\sqrt{10}} - j\frac{1}{\sqrt{10}}$ $13: +\frac{1}{\sqrt{10}} - j\frac{1}{\sqrt{10}}$ $14: -\frac{3}{\sqrt{10}} - j\frac{3}{\sqrt{10}}$ $15: -\frac{1}{\sqrt{10}} + j\frac{1}{\sqrt{10}}$	
Notes:			From [13]	

Label	C.L. (sym)	Tot Mem (bits)	Generator Sequences (octal)
cc4_5.tcm	2	3	4 0 0 0 0 0 4 0 0 0 0 0 4 2 0 0 0 3 5 2
Mapping:	$0: +\frac{3}{\sqrt{20}} - j\frac{1}{\sqrt{20}}$	$16: -\frac{5}{\sqrt{20}} - j\frac{1}{\sqrt{20}}$	
	$1: +\frac{1}{\sqrt{20}} + j\frac{3}{\sqrt{20}}$	$17: +\frac{1}{\sqrt{20}} - j\frac{5}{\sqrt{20}}$	
	$2: +\frac{1}{\sqrt{20}} - j\frac{3}{\sqrt{20}}$	$18: +\frac{5}{\sqrt{20}} + j\frac{1}{\sqrt{20}}$	
	$3: +\frac{3}{\sqrt{20}} + j\frac{1}{\sqrt{20}}$	$19: -\frac{1}{\sqrt{20}} + j\frac{5}{\sqrt{20}}$	
	$4: +\frac{3}{\sqrt{20}} + j\frac{3}{\sqrt{20}}$	$20: -\frac{5}{\sqrt{20}} + j\frac{3}{\sqrt{20}}$	
	$5: -\frac{3}{\sqrt{20}} + j\frac{3}{\sqrt{20}}$	$21: -\frac{3}{\sqrt{20}} - j\frac{5}{\sqrt{20}}$	
	$6: +\frac{1}{\sqrt{20}} + j\frac{1}{\sqrt{20}}$	$22: +\frac{5}{\sqrt{20}} - j\frac{3}{\sqrt{20}}$	
	$7: +\frac{3}{\sqrt{20}} - j\frac{3}{\sqrt{20}}$	$23: +\frac{3}{\sqrt{20}} + j\frac{5}{\sqrt{20}}$	
	$8: -\frac{1}{\sqrt{20}} + j\frac{3}{\sqrt{20}}$	$24: -\frac{1}{\sqrt{20}} - j\frac{5}{\sqrt{20}}$	
	$9: -\frac{3}{\sqrt{20}} - j\frac{1}{\sqrt{20}}$	$25: +\frac{5}{\sqrt{20}} - j\frac{1}{\sqrt{20}}$	
	$10: -\frac{3}{\sqrt{20}} + j\frac{1}{\sqrt{20}}$	$26: +\frac{1}{\sqrt{20}} + j\frac{5}{\sqrt{20}}$	
	$11: -\frac{1}{\sqrt{20}} - j\frac{3}{\sqrt{20}}$	$27: -\frac{5}{\sqrt{20}} + j\frac{1}{\sqrt{20}}$	
	$12: -\frac{1}{\sqrt{20}} - j\frac{1}{\sqrt{20}}$	$28: +\frac{3}{\sqrt{20}} - j\frac{5}{\sqrt{20}}$	
	$13: +\frac{1}{\sqrt{20}} - j\frac{1}{\sqrt{20}}$	$29: +\frac{5}{\sqrt{20}} + j\frac{3}{\sqrt{20}}$	
	$14: -\frac{3}{\sqrt{20}} - j\frac{3}{\sqrt{20}}$	$30: -\frac{3}{\sqrt{20}} + j\frac{5}{\sqrt{20}}$	
	$15: -\frac{1}{\sqrt{20}} + j\frac{1}{\sqrt{20}}$	$31: -\frac{5}{\sqrt{20}} - j\frac{3}{\sqrt{20}}$	
Notes:	From [13]		

B.3 ISI Channels

Two Tap ISI Channels

Label	Chan. mem.	c_o	c_1
isi2.50.50	1	$\sqrt{0.5}$	$\sqrt{0.5}$
isi2.60.40	1	$\sqrt{0.6}$	$\sqrt{0.4}$
isi2.70.30	1	$\sqrt{0.7}$	$\sqrt{0.3}$
isi2.80.20	1	$\sqrt{0.8}$	$\sqrt{0.3}$
isi2.90.10	1	$\sqrt{0.9}$	$\sqrt{0.1}$

Three Tap ISI Channels

Label	Chan. mem.	c_o	c_1	c_2
isi3.60.20.20	2	$\sqrt{0.6}$	$\sqrt{0.2}$	$\sqrt{0.2}$
isi3.80.10.10	2	$\sqrt{0.8}$	$\sqrt{0.1}$	$\sqrt{0.1}$
isi3.100.0.0	2	$\sqrt{1.0}$	$\sqrt{0.0}$	$\sqrt{0.0}$

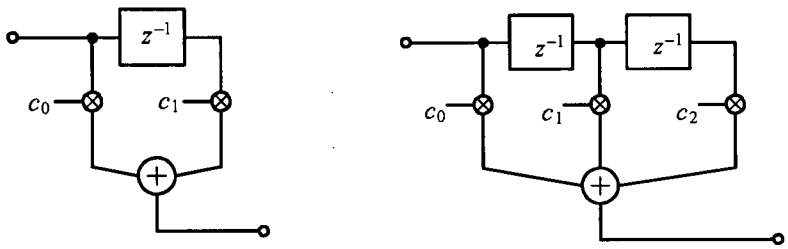


Figure B.1: Two and Three Tap ISI Channels.

Appendix C

The MAP and "Best-Path" Algorithms

The following descriptions of the MAP and "best-path" algorithms are based on the derivations presented in the paper written by Humblet [7].

C.1 The MAP (or Bahl, "forward-backward", "any path") Algorithm

Consider a discrete-time finite-state Markovian process. An input sequence $X_1^N = (X_1, X_2, ..., X_N)$ will result in a state sequence $S_1^{N+1} = (S_1, S_2, ..., S_{N+1})$ along with an associated output sequence. It is assumed that the starting state S_1 and final state S_{N+1} are known (The final input symbols are preset such that the process terminates in a known state.). Transmission of the output sequence over a discrete-time memoryless channel will result in the sequence $Y_1^N = (Y_1, Y_2, ..., Y_N)$ being observed by the receiver. Upon detection of the sequence Y_1^N, the MAP receiver calculates the a posteriori state probabilities $P(S_n|Y_1^N)$ or transitional probabilities $P(S_n^{n+1}|Y_1^N)$. From these, it is possible to deduce the a posteriori input symbol probabilities $P(X_n|Y_1^N)$. Using the a posteriori input symbol probabilities the receiver may make a decoding decision or, in the case of a soft-output decoder, output the a posteriori probabilities themselves.

An explanation of how the MAP algorithm calculates the state/transition probabilities begins by taking into consideration the following probability expansion:

$$P(S_1^{N+1}, Y_1^N) = \overbrace{\overbrace{\overbrace{P(S_1)P(S_2, Y_1|S_1)}^{P(S_1^2, Y_1)} P(S_3, Y_2|S_1^2, Y_1)}^{P(S_1^3, Y_1^2)} P(S_4, Y_3|S_1^3, Y_1^2) \cdots P(S_n, Y_{n-1}|S_1^{n-1}, Y_1^{n-2})}^{P(S_1^n, Y_1^{n-1})} \times$$
$$\underbrace{\underbrace{\underbrace{P(S_{n+1}, Y_n|S_1^n, Y_1^{n-1}) \cdots P(S_{N-1}, Y_{N-2}|S_1^{N-2}, Y_1^{N-3}) P(S_N, Y_{N-1}|S_1^{N-1}, Y_1^{N-2})}_{P(S_{N-1}^{N+1}, Y_{N-2}^N|S_1^{N-2}, Y_1^{N-3})} P(S_{N+1}, Y_N|S_1^N, Y_1^{N-1})}_{P(S_N^{N+1}, Y_{N-1}^N|S_1^{N-1}, Y_1^{N-2})}}_{P(S_{n+1}^{N+1}, Y_n^N|S_1^n, Y_1^{n-1})} \tag{69}$$

Each of the terms on the right hand side of equation (69) can be simplified by noting the following property of Markovian processes: given S_n and S_{n+1}, Y_n is independent of all previous states and received samples. Similarly, given S_n, S_{n+1} is also independent of the same previous states and received samples. Therefore,

$$\begin{aligned} P(S_{n+1}, Y_n|S_1^n, Y_1^{n-1}) &= P(S_{n+1}|S_1^n, Y_1^{n-1}) P(Y_n|S_{n+1}, S_1^n, Y_1^{n-1}) \\ &= P(S_{n+1}|S_n) P(Y_n|S_{n+1}, S_n) \\ &= P(S_{n+1}, Y_n|S_n) \end{aligned} \tag{70}$$

Similarly,

$$P(S_{n+1}^{N+1}, Y_n^N|S_1^n, Y_1^{n-1}) = P(S_{n+1}^{N+1}, Y_n^N|S_n) \tag{71}$$

It is convenient to define the following quantity:

$$Q(Y_n, S_n, S_{n+1}) \triangleq P(S_{n+1}, Y_n|S_n)$$

The first of two recursive relations is now noted:

$$\begin{aligned} P(S_n, Y_1^{n-1}) &= \sum_{s_1^{n-1} \in \mathcal{S}^{n-1}} P(S_1^n, Y_1^{n-1}) \qquad \longleftarrow \text{"upper-half" of eqn (69)} \\ &= \sum_{s_1^{n-1} \in \mathcal{S}^{n-1}} P(S_1^{n-1}, Y_1^{n-2}) P(S_n, Y_{n-1}|S_1^{n-1}, Y_1^{n-2}) \end{aligned}$$

$$= \sum_{s_{n-1}\in\mathcal{S}} \left[\sum_{s_1^{n-2}\in\mathcal{S}^{n-2}} P(S_1^{n-1}, Y_1^{n-2}) \right] P(S_n, Y_{n-1}|S_{n-1})$$
$$= \sum_{s_{n-1}\in\mathcal{S}} P(S_{n-1}, Y_1^{n-2}) P(S_n, Y_{n-1}|S_{n-1}) \tag{72}$$

Defining $P(S_n, Y_1^{n-1})$ as the "forward" probability $P^f(S_n, Y_1^{n-1})$ gives:

$$P^f(S_n, Y_1^{n-1}) = \sum_{s_{n-1}\in\mathcal{S}} P^f(S_{n-1}, Y_1^{n-2}) Q(Y_{n-1}, S_{n-1}, S_n) \tag{73}$$

Since it was assumed that the Markovian process starts in a known state, for instance s_α, the calculation of equation (73) begins with the initial condition:

$$P^f(S_1) = \begin{cases} 1 & , \quad S_1 = s_\alpha \\ 0 & , \quad S_1 \neq s_\alpha \end{cases} \tag{74}$$

The second recursive relation can be found by noting that:

$$\begin{aligned} P(Y_n^N|S_n) &= \sum_{S_{n+1}^{N+1}\in\mathcal{S}^{N-n+1}} P(S_{n+1}^{N+1}, Y_n^N|S_n) \\ &= \sum_{S_{n+1}^{N+1}\in\mathcal{S}^{N-n+1}} P(S_{n+1}^{N+1}, Y_n^N|S_1^n, Y_1^{n-1}) \qquad \longleftarrow \text{"lower-half" of eqn (69)} \\ &= \sum_{S_{n+1}^{N+1}\in\mathcal{S}^{N-n+1}} P(S_{n+1}, Y_n|S_1^n, Y_1^{n-1}) P(S_{n+2}^{N+1}, Y_{n+1}^N|S_1^{n+1}, Y_1^n) \\ &= \sum_{S_{n+1}\in\mathcal{S}} P(S_{n+1}, Y_n|S_n) \sum_{S_{n+2}^{N+1}\in\mathcal{S}^{N-n}} P(S_{n+2}^{N+1}, Y_{n+1}^N|S_{n+1}) \\ &= \sum_{S_{n+1}\in\mathcal{S}} P(S_{n+1}, Y_n|S_n) P(Y_{n+1}^N|S_{n+1}) \end{aligned} \tag{75}$$

Defining $P(Y_n^N|S_n)$ as the "backward" probability $P^b(S_n, Y_n^N)$ yields:

$$P^b(S_n, Y_n^N) = \sum_{S_{n+1}\in\mathcal{S}} Q(Y_n, S_n, S_{n+1}) P^b(S_{n+1}, Y_{n+1}^N) \tag{76}$$

If s_ω represents the assumed terminating state, then the initial condition is:

$$P^b(S_{N+1}) = \begin{cases} 1 & , \quad S_{N+1} = s_\omega \\ 0 & , \quad S_{N+1} \neq s_\omega \end{cases} \tag{77}$$

Once the "forward" and "backward" probabilities have been calculated the state probabilities can be determined by:

$$\begin{aligned} P(S_n, Y_1^N) &= \sum_{S_1^{n-1} \in \mathcal{S}^{n-1}} \sum_{S_{n+1}^{N+1} \in \mathcal{S}^{N-n+1}} P(S_1^{N+1}, Y_1^N) \\ &= \sum_{S_1^{n-1} \in \mathcal{S}^{n-1}} \sum_{S_{n+1}^{N+1} \in \mathcal{S}^{N-n+1}} P(S_1^n, Y_1^{n-1}) P(S_{n+1}^{N+1}, Y_n^N | S_1^n, Y_1^{n-1}) \\ &= \sum_{S_1^{n-1} \in \mathcal{S}^{n-1}} P(S_1^n, Y_1^{n-1}) \sum_{S_{n+1}^{N+1} \in \mathcal{S}^{N-n+1}} P(S_{n+1}^{N+1}, Y_n^N | S_n) \\ &= P^f(S_n, Y_1^{n-1}) P^b(S_n, Y_n^N) \end{aligned} \tag{78}$$

The conditional state probability may be found by:

$$P(S_n | Y_1^N) = \frac{P(S_n, Y_1^N)}{P(Y_1^N)} \tag{79}$$

The transitional probabilities are found from:

$$\begin{aligned} P(S_n^{n+1}, Y_1^N) &= \sum_{S_1^{n-1} \in \mathcal{S}^{n-1}} \sum_{S_{n+2}^{N+1} \in \mathcal{S}^{N-n}} P(S_1^{N+1}, Y_1^N) \\ &= \sum_{S_1^{n-1} \in \mathcal{S}^{n-1}} \sum_{S_{n+2}^{N+1} \in \mathcal{S}^{N-n}} P(S_1^n, Y_1^{n-1}) P(S_{n+1}^{N+1}, Y_n^N | S_1^n, Y_1^{n-1}) \\ &= \sum_{S_1^{n-1} \in \mathcal{S}^{n-1}} \sum_{S_{n+2}^{N+1} \in \mathcal{S}^{N-n}} P(S_1^n, Y_1^{n-1}) P(S_{n+1}, Y_n | S_1^n, Y_1^{n-1}) P(S_{n+2}^{N+1}, Y_{n+1}^N | S_1^{n+1}, Y_1^n) \\ &= \left[\sum_{S_1^{n-1} \in \mathcal{S}^{n-1}} P(S_1^n, Y_1^{n-1}) \right] \left[P(S_{n+1}, Y_n | S_n) \right] \left[\sum_{S_{n+2}^{N+1} \in \mathcal{S}^{N-n}} P(S_{n+2}^{N+1}, Y_{n+1}^N | S_{n+1}) \right] \\ &= P^f(S_n, Y_1^{n-1}) Q(Y_n, S_n, S_{n+1}) P^b(S_{n+1}, Y_{n+1}^N) \end{aligned} \tag{80}$$

The conditional transition probability may be found by:

$$P(S_n^{n+1} | Y_1^N) = \frac{P(S_n^{n+1}, Y_1^N)}{P(Y_1^N)} \tag{81}$$

Finally, to find the a posteriori input symbol probabilities:

$$P(X_n, Y_1^N) = \sum_{S_n^{n+1} | \mathcal{X}(S_n, S_{n+1}) = X_n} P(S_n^{n+1}, Y_1^N) \tag{82}$$

where $\mathcal{X}(S_n, S_{n+1})$ is a mapping from the state transition to the associated input symbol.

C.2 The "Best-Path" Algorithm

The "best-path" algorithm comes about by noting that the Viterbi algorithm uses the iteration:

$$\begin{aligned}\tilde{P}^f(S_n, Y_1^{n-1}) &\triangleq \max_{S_1^{n-1}\in\mathcal{S}^{n-1}} P(S_1^n, Y_1^{n-1}) \\ &= \max_{s_{n-1}\in\mathcal{S}}\left[\max_{s_1^{n-2}\in\mathcal{S}^{n-2}} P(S_1^{n-1}, Y_1^{n-2})\right] P(S_n, Y_{n-1}|S_{n-1}) \\ &= \max_{s_{n-1}\in\mathcal{S}} \tilde{P}^f(S_{n-1}, Y_1^{n-2}) Q(Y_{n-1}, S_{n-1}, S_n) \end{aligned} \tag{83}$$

where $\tilde{P}^f(S_{n-1}, Y_1^{n-2})$ represent the survivor path metrics to each previous state and $Q(Y_{n-1}, S_{n-1}, S_n)$ are the branch metrics associated with each possible transition.

In a similar manner, maximizing in the reverse direction:

$$\begin{aligned}\tilde{P}^b(S_n, Y_n^N) &\triangleq \max_{S_{n+1}^{N+1}\in\mathcal{S}^{N-n+1}} P(S_{n+1}^{N+1}, Y_n^N|S_n) \\ &= \max_{S_{n+1}^{N+1}\in\mathcal{S}^{N-n+1}} P(S_{n+1}, Y_n|S_1^n, Y_1^{n-1}) P(S_{n+2}^{N+1}, Y_{n+1}^N|S_1^{n+1}, Y_1^n) \\ &= \max_{S_{n+1}\in\mathcal{S}} P(S_{n+1}, Y_n|S_n) \max_{S_{n+2}^{N+1}\in\mathcal{S}^{N-n}} P(S_{n+2}^{N+1}, Y_{n+1}^N|S_{n+1}) \\ &= \max_{S_{n+1}\in\mathcal{S}} Q(Y_n, S_n, S_{n+1}) \tilde{P}^b(S_{n+1}, Y_{n+1}^N) \end{aligned} \tag{84}$$

This formula once again describes the Viterbi algorithm except it is now run in reverse. $\tilde{P}^b(S_{n+1}, Y_{n+1}^N)$ are the reverse path metrics and $Q(Y_{n-1}, S_{n-1}, S_n)$ are, as stated previously, the branch metrics.

Now that the best "forward" and "backward" probabilities to each state are known, the probability of the best path through a particular state can be determined by:

$$\begin{aligned}\tilde{P}(S_n, Y_1^N) &\triangleq \max_{S_1^{n-1}\in\mathcal{S}^{n-1}} \max_{S_{n+1}^{N+1}\in\mathcal{S}^{N-n+1}} P(S_1^{N+1}, Y_1^N) \\ &= \tilde{P}^f(S_n, Y_n^{n-1}) \tilde{P}^b(S_n, Y_n^N) \end{aligned} \tag{85}$$

Likewise, the probability of the best path with a specific transition may be found by:

$$\begin{aligned}\tilde{P}(S_n^{n+1}, Y_1^N) &\triangleq \max_{S_1^{n-1}\in\mathcal{S}^{n-1}} \max_{S_{n+2}^{N+1}\in\mathcal{S}^{N-n}} P(S_1^{N+1}, Y_1^N) \\ &= \tilde{P}^f(S_n, Y_1^{n-1}) Q(Y_n, S_n, S_{n+1}) \tilde{P}^b(S_{n+1}, Y_{n+1}^N) \qquad (86)\end{aligned}$$

By using the probabilities determined in equation (86) in equation (82) one is able to obtain an estimate of the a posteriori input symbol probabilities.

C.3 Computational and Storage Requirements

A decoder can be implemented to be efficient in either the amount of computation required or the amount of storage space required. In the discussion that follows, for the determination of the minimum storage requirements, the "save storage" implementation shall be assumed, whereas, for the calculation of the minimum computational requirements, the "save time" implementation shall be used. It should be pointed out that in the case of the MAP and Best-Path algorithms, these two cost factors are mutually exclusive. However, the resulting expressions for computational and storage requirements will be the best that can be achieved by the basic MAP and Best-Path algorithms. It is with these best case metrics that the comparisons with other competing algorithms shall be made.

Consider decoding a sequence of N received symbols produced by the transmission of N output symbols from an (n, k, m) Markovian process. Assuming a minimum memory configuration of either the MAP or Best-Path algorithms, the decoder will need to store the received sequence so that it may calculate $Q(Y_n, S_n, S_{n+1})$ (a quantity that is similar to the Viterbi branch metric) as needed. This will require the allocation of N storage elements (double this if the received sequence is complex). If it is desired that the output information come out of the decoder in the correct order without having to

buffer decisions, it will be necessary to calculate and store $P^b(S_n, Y_n^N)$. This will require the allocation of $N2^m$ storage locations. The subsequent calculation of $P^f(S_n, Y_1^{n-1})$ can then be made immediately followed by the calculation of the transition probabilities $P(S_n^{n+1}, Y_1^N)$. If the received sequence is sufficiently long, the amount of temporary storage required to calculate these quantities will be small in comparison to the storage requirements of the received sequence and $P^b(S_n, Y_n^N)$. Hence a minimum memory configuration decoder will require at least $N2^m + N$ storage elements.

To determine the computational requirements of a typical MAP decoder, the assumption of a "save time" implementation is made. (In this implementation the quantities $Q(Y_n, S_n, S_{n+1})$ are calculated once only and stored for later retrieval as needed.) Calculation of probabilities $P^f(S_n, Y_1^{n-1})$ and $P^b(S_n, Y_n^N)$ will each require: $2^m 2^k$ binary multiplications and $2^m(2^k - 1)$ binary additions per symbol. $Q(Y_n, S_n, S_{n+1})$ requires essentially the same computational effort as the Viterbi metric with the exception that an additional exponential must be calculated to return $e^{-metric}$. Since this quantity depends on the specific application it is not possible to determine a general expression for the amount of elementary operations required for its computation. However since a metric is calculated for each possible output symbol at each time index it is safe to assume that the time required will be proportional to 2^n per symbol. The transition probabilities will each require 2 binary multiplications for each of the $2^m 2^k$ state transitions per time index for a total of $2^m 2^{k+1}$ binary multiplications per symbol. If the effort required to calculate the input symbol probability is neglected (which consists of several compares, additions, and normalization if required) then, excluding the computation required for $Q(Y_n, S_n, S_{n+1})$, the total amount of elementary operations is as shown in table C.1.

When implementing the "best-path" algorithm, one usually uses the logarithm of the quantities $\tilde{P}^f(S_n, Y_1^{n-1})$, $\tilde{P}^b(S_n, Y_n^N)$, and $Q(Y_n, S_n, S_{n+1})$. Therefore, for a "save time" implementation of the "best-path" algorithm, the following calculations will have to be

	binary multiplications	binary additions	exp
$P^f(S_n, Y_1^{n-1})$	$2^m 2^k$	$2^m(2^k-1)$	
$P^b(S_n, Y_n^N)$	$2^m 2^k$	$2^m(2^k-1)$	
$Q(Y_n, S_n, S_{n+1})$	(see text)		2^n
$P(S_n^{n+1}, Y_1^N)$	$2^m 2^{k+1}$		

Table C.1: Minimum Operations Required per Input Symbol for the MAP Algorithm.

	binary compares	binary additions	exp
$\log \tilde{P}^f(S_n, Y_1^{n-1})$	$2^m(2^k-1)$	$2^m 2^k$	
$\log \tilde{P}^b(S_n, Y_n^N)$	$2^m(2^k-1)$	$2^m 2^k$	
$\log Q(Y_n, S_n, S_{n+1})$	(see text)		
$\tilde{P}(S_n^{n+1}, Y_1^N)$		$2^m 2^{k+1}$	$2^m 2^k$

Table C.2: Minimum Operations Required per Input Symbol for the Best-Path Algorithm.

made. The determination of $\log(\tilde{P}^f(S_n, Y_1^{n-1}))$ and $\log(\tilde{P}^b(S_n, Y_n^N))$ will each require $2^m 2^k$ binary additions and $2^m(2^k-1)$ binary compares per symbol. Computation of $\log(Q(Y_n, S_n, S_{n+1}))$ does not require any exponentials however exponentials are now required during the later step when the trellis transition probabilities are calculated. The transition probabilities will require 2 binary additions for each of the $2^m 2^k$ state transitions for a total of $2^m 2^{k+1}$ binary additions per symbol. The total number of operations are summarized in table C.2.