JOINT SOURCE-CHANNEL DECODING OF MULTIPLE DESCRIPTIONS

by

Upul Samarawickrama

A Thesis
Submitted to the Faculty of Graduate Studies
In Partial Fulfillment of the Requirements
For the Degree of
Master of Science

Department of Electrical and Computer Engineering
University of Manitoba
Winnipeg, Manitoba, Canada

THE UNIVERSITY OF MANITOBA

FACULTY OF GRADUATE STUDIES

*****

COPYRIGHT PERMISSION

Joint Source-Channel Decoding of Multiple Descriptions

by

Upul Samarawickrama

A Thesis/Practicum submitted to the Faculty of Graduate Studies of The University of

Manitoba in partial fulfillment of the requirement of the degree

of

Master of Science

Abstract

Multiple description coding is an emerging source coding based solution for packet loss problem in data communication networks. Transmission over a practical network also requires such multiple descriptions to be channel coded to guard against random bit errors caused by channel noise. In this context, the redundancy introduced by both multiple description source encoder and the channel encoder can be exploited in a joint source-channel decoder to improve the overall reliability of the system.

The main objective of this thesis is to investigate the problem of joint source-channel decoding in a system based on multiple description quantization and to develop efficient decoding schemes. A review of the previously proposed joint decoding schemes is given with an experimental evaluation of their performance. Subsequently, a system with two channel multiple description quantization is considered in which the output of a quantizer is convolutional encoded prior to transmission. Based on the idea of concatenated coding, a joint decoding scheme that uses list Viterbi algorithm is presented. Experimental results show that this scheme is capable of achieving a significant improvement in the error correcting capability of the convolutional code at moderate complexities. A previously proposed optimum trellis decoding scheme which performs decoding at symbol level is extended to a bit level decoding scheme. It is shown that bit level decoding in this manner considerably reduces the computational complexity. While this scheme is in general not optimal, it is also shown that the scheme is indeed optimal when the convolutional encoder memory is greater than a particular parameter related to both the number of bits used per description and the temporal correlation of the multiple description quantizer output. Finally, a preliminary investigation of the problem of designing a joint source-channel encoder in a multiple description communication system is presented. In this context, a joint encoding procedure which involves re-mapping the indices of a multiple description quantizer output to improve the distance properties of a channel code is presented. It is shown experimentally that this joint encoding procedure considerably improves the performance of the system.

Acknowledgements

I would like to express my sincere appreciation and gratitude to Professor P. Yahampath for his continuous guidance, encouragement and support throughout the course of this work. His advice and assistance in the preparation of this thesis is thankfully appreciated. Appreciation is also extended to the members of my thesis examining committee, Professor E. Shwedyk and Professor A. Thavaneswaran, for graciously accepting to review the thesis.

I would like to thank all my friends and the staff of the Department of Electrical and Computer Engineering for their invaluable assistance in this research work.

I extend my heartfelt gratitude to my wife and to my parents for their encouragement and moral support. I would also like to thank my brothers and my sister for all the love and support.

Upul Samarawickrama

August 2005

To my parents and teachers

Table of Contents

List of Figures

List of Tables

Acronyms

BLJT	Bit level joint trellis
BSC	Binary symmetric channel
DLJT	Description level joint trellis
GM	Gauss Markov
IA	Index assignment
i.i.d.	Independently and identically distributed
JPM	Joint path metric
JPM-MAP	Joint path metric maximum a posteriori probability
JPM-ML	Joint path metric maximum likelihood
JT	Joint trellis
JT-MAP	Joint trellis maximum a posteriori probability
JT-ML	Joint trellis maximum likelihood
LVA	List viterbi algorithm
MAP	Maximum a posteriori probability
MD	Multiple description
MDC	Multiple description coding
MDQ	Multiple description quantization
MDSQ	Multiple description scalar quantization
MDVQ	Multiple description vector quantization
ML	Maximum likelihood
MSE	Mean square error
PPM	Partial path metric
SDR	Signal to distortion ratio
SOVA	Soft output Viterbi algorithm
SPM	Separate path metric
SQ	scalar quantizer
VA	Viterbi algorithm
VQ	Vector quantizer

Chapter 1

Introduction

Transmission of information over a digital communication system in general involves four basic operations, namely, source coding, channel coding, modulation and transmission as depicted in Figure 1.1. The input to the source encoder can be either an analog or digital signal. The source encoder generates an efficient digital representation of the input signal usually in binary format. The main objective of source encoding is to obtain a representation with as few symbols as possible (compression), which is done by removing the redundancy present in the input signal. Source coding can be mainly categorized into two types as lossy and lossless coding. The lossless coding preserves all the information in the input signal while, as its name implies, lossy coding involves a loss of some information. The function of the channel encoder is to add redundancy to its input in a controlled manner to achieve robustness against the errors caused in the transmission channel. The added redundancy helps the channel decoder to recover reliably the transmitted data. The channel usually carries the information as an analog signal and hence it is needed to map the digital signal at the channel encoder output into an analog signal, which is done by the modulator. The function of demodulator, channel decoder and the source decoder is to estimate the digital signal at the input of the respective encoder at the sending end. In some digital communication systems, data is transmitted in blocks called *packets* or *frames*. Data communication networks, such as the Internet suffer from the problem of packet losses caused by congestion

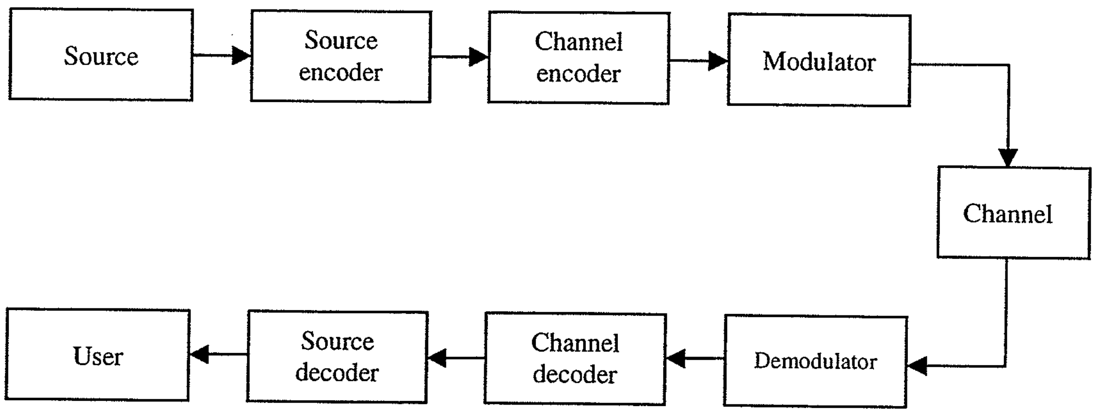


Figure 1.1: A digital communication system.

in the network. In some situations, this problem can be solved by retransmission of lost packets. However, this requires a feedback path to request retransmission and therefore cannot be used in broadcast type communications. Retransmission may also not be possible in some real time communication systems (such as in audio and video communication), where the allowable delay is limited. In case of transmission of an analog source over a packet loss network, an emerging source coding based solution for the above problem is Multiple description coding (MDC) where, several versions referred to as "descriptions" of the same message are generated and transmitted such that, each describes the message with a certain acceptable quality, and when more than one description is available at the receiver they can be combined to reconstruct the message at a higher quality. The term "message" may refer to one source sample or a group of source samples. Different descriptions of the same message are transmitted over separate communication paths and hence it is less likely to lose all of them simultaneously. Therefore, with this method a certain minimum quality of service can be guaranteed at the receiver. Another application of MDC suggested in literature is in wireless mobile networks. In wireless mobile communication, the transmitted signal arrives at the receiver over multiple paths. These *multipath* signals may, from time to time, interfere destructively with each other, causing a weak signal level at the receiver. This phenomenon is known as multipath *fading*. A frame under deep fading can be considered as lost and therefore multiple description coding can be used in this situation as well. Recently, a significant amount of research has been done in MDC and many methods have been developed to generate multiple descriptions from a source.

MDC, as explained in the next section, is mainly a source coding method since it performs a lossy compression of the analog source signal. On the other hand, it also involves a systematic addition of redundancy, which is a feature of channel coding to achieve reliability over channel losses. Hereafter, we use the term *channel erasure* to refer to a channel loss. On the other hand, the term *channel error* is used to refer to the random bit errors in received symbols caused by channel noise. A coding scheme, which performs both channel coding and source coding jointly, is known as a joint source-channel code and hence MDC has some features of joint source-channel coding. However, in MDC, the redundancy is added to guard against channel erasures and therefore it alone is not sufficient for correcting the channel errors. Hence, in practice, it is required to use channel coding after MDC to achieve reliability over channel errors. In this case, it would be possible to device a channel decoding scheme which exploits not only the redundancy added by the channel coder but also that added by the MD encoder to enhance the error correcting capability. We call such a decoding scheme as *joint source-channel decoding of multiple descriptions.* This chapter gives an introduction to MDC and joint source-channel decoding of multiple descriptions.

1.1 Multiple Description Coding

MDC was invented in the 1970s as a speech coding method for achieving reliability in telephone networks. When a link fails in a telephone network the calls have to be diverted to a standby link. These standby links add extra cost to the system. On the other hand they are not used in the normal operation and therefore can be considered as a waste of resources. The number of standby links can be greatly reduced if the information in a call can be split into two streams and sent over two channels such that each independently gives an acceptable quality. Jayant [1] developed odd-even sample separation based speech coding techniques where the odd numbered voice samples and even numbered voice samples are send over two different channels. When both the channels work the speech is reconstructed at a higher quality. On the other hand, when one fails, samples on the other channel are used to reconstruct the voice at a lower quality. MDC was studied as an information theoretical problem in late 1970s and early 1980s by several authors. El Gamal et al. in [2] provided a rate distortion analysis of the MDC problem for a memoryless source and constructed the achievable rate region. Ozarow [3] found the achievable rate region for a memoryless Gaussian source for square error distortion measure and showed that it is the largest set that can be obtained with the rate region derived in [2]. Some of Ozarow's results will be discussed at the end of this section. A good review on MDC can be found in [4].

Not much work was done in MDC until Vaishampayan [5] in 1993 provided the first theoretical framework for the practical implementation of multiple description scalar quantizers (MDSQ). His method of MD generation consists of scalar quantization followed by an *index assignment* (IA). MDSQ will be discussed in more details in Section 1.1.1. Since Vaishampayan's initial work on MDSQ in 1993, several methods of MD coding have been developed. Wang et al. [6] proposed a correlating transform based MDC technique where a linear transform is applied on source vectors (an ordered tuple of source samples) to generate correlated transform coefficients. This method is based on the fact that, when some of the coefficients are lost, they can be estimated from the available coefficients by exploiting the correlation introduced by the linear transform. Goyal et al. [7] presented another transform coding based MD coding method which uses the concept of frame expansion to generate MD.

1.1.1 Multiple Description Scalar Quantization

In this section, we discuss in detail the MDSQ presented in [5]. Figure 1.2 shows two channel MDSQ of a random variable X that has the probability density $p(x)$. Assume that the channels can deliver information at rates R_1 and R_2 bits/source sample. Each channel can be in one of two states. In one of the states, the channel delivers the information error free while in the other state a complete loss of information (erasure) occurs. It is assumed

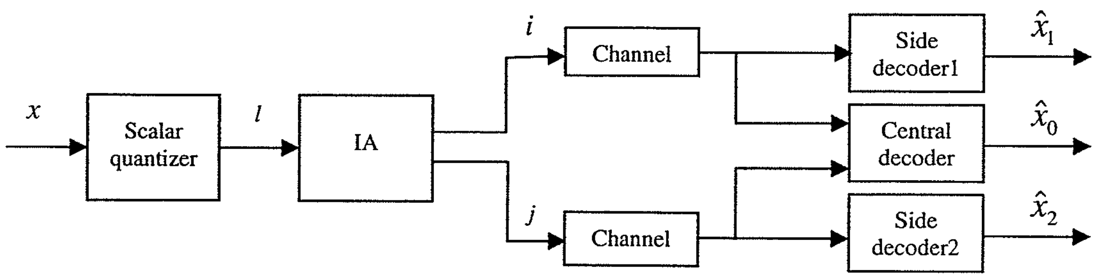


Figure 1.2: Multiple description scaler quantization.

that the states of the channels are known to the decoder but not to the encoder. A two channel MDSQ consists of

- a scalar quantizer (SQ) $f : \mathbb{R} \to \mathcal{Q} = \{1, 2, \dots, N\}$
- an index assignment (IA) $a : \mathcal{Q} \to \mathcal{I} \times \mathcal{J}$ where $\mathcal{I} = \{1, 2, \dots, M_1\}$, $\mathcal{J} = \{1, 2, \dots, M_2\}$, $M_1 \leq 2^{R_1}$, $M_2 \leq 2^{R_2}$ and the mapping a is one-to-one
- two side decoders $g_1 : \mathcal{I} \to \mathcal{C}_1 = \{\hat{x}_1(1), \hat{x}_1(2), \dots, \hat{x}_1(M_1)\}$ and $g_2 : \mathcal{J} \to \mathcal{C}_2 = \{\hat{x}_2(1), \hat{x}_2(2) \dots, \hat{x}_2(M_2)\}$, where $\mathcal{C}_1$ and $\mathcal{C}_2$ are the *side codebooks*
- the central decoder $g_0 : \mathcal{S} \to \mathcal{C}_0 = \{\hat{x}_0(1), \hat{x}_0(2), \dots, \hat{x}_0(N)\}$ where $\mathcal{S} = \{(i, j) : (i, j) = a(l), l \in \mathcal{Q}\}$ and $\mathcal{C}_0$ is the *central codebook*

It should be noted that N may be less than $(M_1 \times M_2)$ and therefore some index pairs (i, j) may not be assigned by the IA.

For each source sample x, the SQ generates an index $l \in \mathcal{Q}$ which is then assigned to the indices (or descriptions) $i \in \mathcal{I}$ and $j \in \mathcal{J}$ by the IA. The descriptions are sent over two channels separately. When only one description is received, the corresponding side decoder outputs an estimate ($\hat{x}_1(i)$ or $\hat{x}_2(j)$ depending on which description is received) of the source sample x. On the other hand, when both descriptions are received, the central decoder outputs a better estimate $\hat{x}_0(l)$. In the case where both descriptions are lost, it is needed to use some other means such as using a pre-determined value as the output or requesting a retransmission.

The SQ partitions the input space into cells $A^0_{ij} = \{x : a(f(x)) = (i,j)\}$ where

$$\bigcup_{ij} A_{ij} = \mathbb{R}$$

and

$$A_{ij} \bigcap A_{lm} = \emptyset \qquad \forall\, (l,m) \neq (i,j).$$

The MDSQ is completely determined by the encoder partition, IA and the decoder codebooks. Let $d(x,\hat{x})$ denote the per letter distortion between the source sample x and its estimate $\hat{x}$. Then, the average distortion D_0 at the output of the central decoder is given by

$$D_0 = \sum_{(i,j)\in\mathcal{S}} \int_{A^0_{ij}} d\big(x, \hat{x}_0(i,j)\big)p(x)\; dx \tag{1.1}$$

where $\hat{x}_0(i,j) \triangleq \hat{x}_0(l)$ and $(i,j) = a^{-1}(l)$. Similarly, the average distortions D_1 and D_2 in the respective side decoders are given by

$$D_1 = \sum_{i\in\mathcal{I}} \int_{A^1_i} d\big(x, \hat{x}_1(i)\big)p(x)\; dx \tag{1.2}$$

and

$$D_2 = \sum_{j\in\mathcal{J}} \int_{A^2_j} d\big(x, \hat{x}_2(j)\big)p(x)\; dx \tag{1.3}$$

where

$$A^1_i = \bigcup_{j\in\mathcal{J}} A^0_{ij}$$

and

$$A^2_j = \bigcup_{i\in\mathcal{I}} A^0_{ij}$$

Let μ_0 be the probability that both descriptions are received by the decoder, μ_m be the probability that only the description m is received, where m =1,2 and μ' be the probability that both descriptions are lost. Let's assume that the decoder output is $\hat{x}'$ when both descriptions are lost. Then, the total average distortion D of the MDSQ system is given by

$$D = \mu_0 D_0 + \mu_1 D_1 + \mu_2 D_2 + \mu' D'$$

where

$$D' = \int_{-\infty}^{\infty} d(x, \hat{x}')p(x)\, dx$$

To find the optimum MDSQ, we have to find the encoder partition, IA and the decoder codebooks that minimizes the total average distortion. However, it is difficult to find a closed form solution to this problem. Instead, the generally used method is to find the optimality requirements of each unit (encoder, IA and the decoder) separately assuming the other units to be fixed and then to apply an iterative descent algorithm such as Lloyd algorithm [8] to find an (possibly local) optimum solution.

Optimum Encoder

Finding the optimum encoder for fixed IA and decoder is equivalent to finding the partition of the input space that minimizes the total average distortion

$$\begin{aligned} D &= \mu_0 \sum_{(i,j)\in\mathcal{S}} \int_{A^0_{ij}} d(x, \hat{x}_0(i,j))p(x)\, dx + \mu_1 \sum_{i\in\mathcal{I}} \int_{A^1_i} d\big(x, \hat{x}_1(i)\big)p(x)\, dx \\ &\quad + \mu_2 \sum_{j\in\mathcal{J}} \int_{A^2_j} d\big(x, \hat{x}_2(j)\big)p(x)\, dx + \mu' \int_{-\infty}^{\infty} d(x, \hat{x}')p(x)\, dx \\ &= \sum_{(i,j)\in\mathcal{S}} \int_{A^0_{ij}} \{\mu_0 d\big(x, \hat{x}_0(i,j)\big) + \mu_1 d\big(x, \hat{x}_1(i)\big) + \mu_2 d\big(x, \hat{x}_2(j)\big) + \mu' d(x, \hat{x}')\}p(x)\, dx \end{aligned} \tag{1.4}$$

It is clear that D is minimized if the term in the curly brackets inside the integration is minimized for each x. Again, the term $d(x, \hat{x}')$ does not depend on the encoder partition and hence does not affect the minimization. Therefore, the optimum encoder partition is given by,

$$\begin{aligned} A_{ij}^0 &= \{x : \mu_0 d\big(x, \hat{x}_0(i,j)\big) + \mu_1 d\big(x, \hat{x}_1(i)\big) + \mu_2 d\big(x, \hat{x}_2(j)\big) \\ &\quad \leq \mu_0 d\big(x, \hat{x}_0(i',j')\big) + \mu_1 d\big(x, \hat{x}_1(i')\big) + \mu_2 d\big(x, \hat{x}_2(j')\big), \forall (i',j') \neq (i,j), (i',j') \in \mathcal{S}\} \end{aligned} \tag{1.5}$$

For example let's consider the square error distortion measure in which the distortion between the source sample x and its estimate $\hat{x}$ is given by

$$d(x, \hat{x}) = (x - \hat{x})^2$$

Using the square error as the distortion measure in (1.5)

$$\begin{aligned} A_{ij}^0 &= \{x : \mu_0 (x - \hat{x}_0(i,j))^2 + \mu_1 (x - \hat{x}_1(i))^2 + \mu_2 (x - \hat{x}_2(j))^2 \\ &\quad \leq \mu_0 (x - \hat{x}_0(i',j'))^2 + \mu_1 (x - \hat{x}_1(i'))^2 + \mu_2 (x - \hat{x}_2(j'))^2, \forall (i',j') \neq (i,j), (i',j') \in \mathcal{S}\} \end{aligned} \tag{1.6}$$

Let

$$\alpha_{ij} = 2(\mu_0 \hat{x}_0(i,j) + \mu_1 \hat{x}_1(i) + \mu_2 \hat{x}_2(j))$$

and

$$\beta_{ij} = \mu_0 \hat{x}_0^2(i,j) + \mu_1 \hat{x}_1^2(i) + \mu_2 \hat{x}_2^2(j) \qquad i \in \mathcal{I} \text{ and } j \in \mathcal{J}$$

Now, (1.6) can be further simplified to

$$A_{ij}^0 = \{x : \alpha_{ij} x - \beta_{ij} \geq \alpha_{i',j'} x - \beta_{i',j'}, \forall (i',j') \neq (i,j), (i',j') \in \mathcal{S}\} \tag{1.7}$$

In this case, it can be shown that each cell A_{ij}^0 in the encoder partition is an interval.

Optimum Decoder

To find the optimum decoder, we have to determine the codebooks $\mathcal{C}_0$, $\mathcal{C}_1$, and $\mathcal{C}_2$ that minimize the total average distortion for a fixed encoder and an IA. From (1.4), it is clear that the minimum D can be achieved by minimizing the distortion in each decoder separately. Re-organizing (1.1)

$$\begin{aligned} D_0 &= \sum_{(i,j)\in\mathcal{S}} P(A_{ij}^0) \int_{A_{ij}^0} d\big(x, \hat{x}_0(i,j)\big) \frac{p(x)}{P(A_{ij}^0)}\, dx \\ &= \sum_{(i,j)\in\mathcal{S}} P(A_{ij}^0) \int_{A_{ij}^0} d\big(x, \hat{x}_0(i,j)\big) p(x|A_{ij}^0)\, dx \qquad (1.8) \end{aligned}$$

where

$$P(A_{ij}^0) = \int_{A_{ij}^0} p(x)\, dx$$

which is the probability that $x \in A_{ij}^0$ and $p(x|A_{ij}^0)$ is the conditional probability density of x given that x is in A_{ij}^0. Therefore, to minimize D_0, it is sufficient to minimize the integration term in (1.8). Hence, the optimum central codebook is given by

$$\begin{aligned} \hat{x}_0(i,j) &= \arg\min_{x'\in\mathbb{R}} \int_{A_{ij}^0} d(x,x') p(x|A_{ij}^0)\, dx \qquad (i,j)\in\mathcal{S} \\ &= \arg\min_{x'\in\mathbb{R}} E(d(x,x')|x \in A_{ij}^0) \qquad (i,j)\in\mathcal{S} \qquad (1.9) \end{aligned}$$

Similarly, the optimum side codebooks are given by

$$\hat{x}_1(i) = \arg\min_{x'\in\mathbb{R}} E(d(x,x')|x \in A_i^1) \qquad i \in \mathcal{I} \qquad (1.10)$$

and

$$\hat{x}_2(j) = \arg\min_{x'\in\mathbb{R}} E(d(x,x')|x \in A_j^2) \qquad j \in \mathcal{J} \qquad (1.11)$$

For square error distortion measure, (1.9), (1.10) and (1.11) can be further simplified as

$$\begin{aligned}\hat{x}_0(i,j) &= \arg\min_{x' \in \mathbb{R}} E((x-x')^2 | x \in A^0_{ij}) \\ &= E(x|x \in A^0_{ij}) \qquad (i,j) \in \mathcal{S} \end{aligned} \tag{1.12}$$

Similarly

$$\hat{x}_1(i) = E(x|x \in A^1_i) \qquad i \in \mathcal{I} \tag{1.13}$$

and

$$\hat{x}_2(j) = E(x|x \in A^2_j) \qquad j \in \mathcal{J} \tag{1.14}$$

For a given IA the MDSQ can be designed using the Lloyd algorithm [8] as follows. First select the initial codebooks. For the given IA and the codebooks find the optimum encoder using (1.5). Now, use this encoder and the given IA to find the optimum decoder using (1.9), (1.10) and (1.11). It should be clear that D must reduce or remain unchanged in each of the two steps. These two steps are repeated until the difference in D in successive iterations is smaller than a preset threshold. A stationary point would be obtained if the iteration is carried out a sufficiently large number of times.

Index Assignment

The IA plays a key role in MDSQ. In this section, we assume that the square error distortion measure is used and therefore the central partition consists of intervals. We further assume that $M_1 = M_2 = M$. To illustrate the IA problem let's consider $R_1 = R_2$ =1 bits/source sample MDSQ of a source uniformly distributed in the range $(-\sqrt{3}, \sqrt{3})$. Figure 1.3 shows 3 possible partitions and IAs. Tables 1.1 and 1.2 give the corresponding optimum codebooks and the distortions. MDSQ in both (a) and (b) use all four index pairs (1,1), (1,2), (2,1) and (2,2) while (c) uses only three out of them. The performance in case (a) is better than that in case (b). However, in both cases, one side decoder has poor performance with distortion close to 1. Here, the source variance is also 1 and any decoder can achieve the

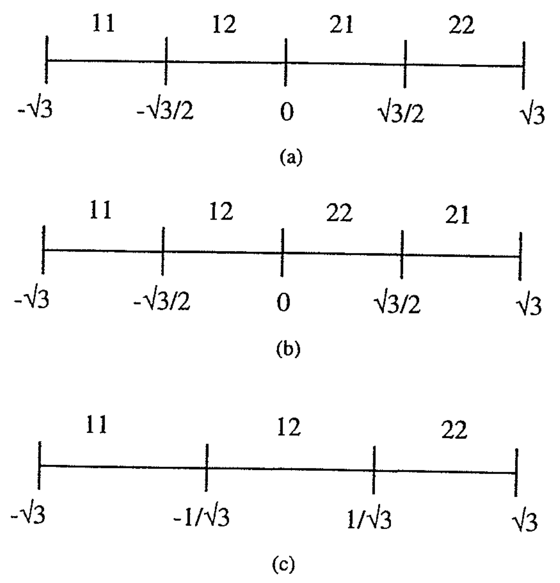


Figure 1.3: Three designs of MDSQ encoder and IA for rate $R_1 = R_2 = 1$ bits/source sample.

Table 1.1: Codebooks of MDSQs in Figure 1.3

	$x_0(1,1)$	$x_0(1,2)$	$x_0(2,1)$	$x_0(2,2)$	$x_1(1)$	$x_1(2)$	$x_2(1)$	$x_2(2)$
(a)	$-3\sqrt{3}/4$	$-\sqrt{3}/4$	$\sqrt{3}/4$	$3\sqrt{3}/4$	$-\sqrt{3}/2$	$\sqrt{3}/2$	$-\sqrt{3}/4$	$\sqrt{3}/4$
(b)	$-3\sqrt{3}/4$	$-\sqrt{3}/4$	$\sqrt{3}/4$	$3\sqrt{3}/4$	$-\sqrt{3}/2$	$\sqrt{3}/2$	0.0	0.0
(c)	$-2/\sqrt{3}$	0.0	—	$2/\sqrt{3}$	$-1/\sqrt{3}$	$2/\sqrt{3}$	$-2/\sqrt{3}$	$1/\sqrt{3}$

distortion equal to the source variance just by using the mean value of the source as the output all the time (i.e. with no information delivered from the source). MDSQ in (c) has an increase in distortion of the central decoder and one of the side decoders. But, it has the desirable feature that both side decoders achieve small distortion (i.e. an effective information delivery has happened to each of the side decoders). From this example, it is clear that the performance of MDSQ greatly depends on IA. The example also suggests that, when all the description pairs are used, at least one side decoder would have very poor performance.

Finding an optimum IA is a basic challenge in the design of an efficient MDSQ . An exhaustive search is impractical for large M since there are $\sum_{N=1}^{M^2}(M^2)!/(M^2-N)!$ possible

Table 1.2: MSE distortions of MDSQs in Figure 1.3

	D_0	D_1	D_2
(a)	1/16	1/4	13/16
(b)	1/16	1/4	1
(c)	1/9	1/3	1/3

IAs to consider and therefore a combinatorial optimization algorithm must be used.

Index assignment can be considered as an assignment of elements in a matrix of dimension $M \times M$ to the cells of the encoder partition. As an example, assigning the index pair (i, j) can be considered as assigning the (i, j)-th element of an $M \times M$ matrix. Let's assume that the intervals in the encoder partition are numbered from left to right in ascending order. When $N < M$, one of the best IAs would be ((1,1), (2,2),..., (N,N)) where the k-th index pair corresponds to the k-th interval in the partition. In this case, the elements assigned in the IA matrix lie in the main diagonal and the side decoders would have the same distortion as the central decoder. When N=M all the elements in the main diagonals are assigned. Suppose that $N = M + 1$ and hence one more index pair has to be assigned. To minimize the increase in side distortion, the next index pair would be of the form (k, k+1) or $(k + 1, k)$ and be assigned to the element next to the one assigned to (k, k). This corresponds to the assignment of an element in a diagonal adjacent to the main diagonal which is illustrated in Figures 1.4 and 1.5 for a MDSQ of a uniform source at rate $R_1 = R_2 = 2$ bits/source sample (i.e. $M = 4$). In this manner, we can argue that the elements on the diagonals closer to the main diagonal have to be assigned before assigning the elements on the diagonals further away from the main diagonal. Vaishampayan [5] has presented two methods for constructing a good IA based on high rate analysis. The two index assignment methods have optimum decay characteristics in the central and side distortions as predicted by rate distortion theory at high rates (i.e. when R_1 and R_2 are sufficiently large). Figure 1.6 shows an IA based on Vaishampayan's nested index assignment for $R_1 = R_2 = 3$ bits/source sample, where 5 diagonals are assigned.

The fraction of the elements assigned in the IA matrix determines the central and side

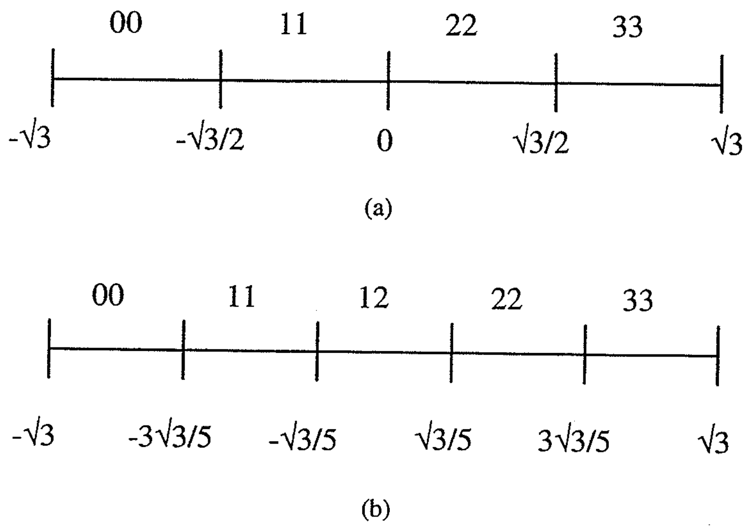


Figure 1.4: Encoder partition and IA for $R_1 = R_2 = 2$ bits/source sample and (a) $N = 4$ (b) $N = 5$.

distortions. When this fraction is increased, the central distortion decreases while the side distortions increase and vice-versa. When the packet loss probability is zero, the end-to-end distortion will be completely determined by the central distortion and therefore the the optimum MDSQ in this case would have all the elements assigned in the IA matrix. In fact, source coding in this case is essentially equivalent to the standard (single description) scalar quantization. As the the packet loss probability is increased, it would be necessary to decrease the number of elements assigned in the IA matrix to obtain the minimum end to end distortion.

1.1.2 Multiple Description Vector Quantization (MDVQ)

A scalar quantizer is a mapping from the input (scalar) space to an index set ($\mathcal{Q}$ in section 1.1.1). Similarly, a vector quantizer is a mapping from the input vector space into an index set [9]. The design of the encoder and the decoder in a MDVQ can be done in essentially the same way as in MDSQ using an iterative algorithm. However, there is no natural ordering in the cells of the encoder partition of MDVQ (remember, in MDSQ, the input space is a subspace of the $\mathbb{R}$ and therefore the encoder cells can be ordered, as an example from

0			
	1		
		2	
			3

(a)

0			
	1	2	
		3	
			4

(b)

Figure 1.5: IA matrices for MDSQ in Figure 1.4

0	1	3					
2	5	6	8				
4	7	10	12	14			
	9	11	15	17	19		
		13	16	20	21	23	
			18	22	25	26	28
				24	27	30	32
					29	31	33

Figure 1.6: Vaishampayan's nested IA for $R_1 = R_2 = 3$ bits/source sample and N=34.

left to right) and therefore Vaishampayan's method of index assignment cannot be applied. Hence, the problem of IA in MDVQ has to be solved with a combinatorial optimization technique such as deterministic annealing [10] and simulated annealing [11]. Hereafter, we use multiple description quantization (MDQ) to refer to both MDSQ and MDVQ.

1.1.3 Rate Distortion Region for MD coding

let's consider two channel MD coding of a source. Let R_1 and R_2 be channel rates and D_0, D_1, and D_2 be central and side distortions respectively. Assume that MD coding is done in vectors of n source samples. A quintuple $(R_1, R_2, D_0, D_1, D_2)$ is said to be achievable if, for some positive integer n, there exist a MD coder with rates R_1 and R_2 and distortions D_0, D_1, and D_2. The rate distortion region for this MD coding is the closure of the set of achievable quintuples $(R_1, R_2, D_0, D_1, D_2)$. Ozarow [3] found the rate distortion region for a memoryless Gaussian source of variance σ^2. In this case, the rate distortion region is given by

$$D_1 \geq \sigma^2 2^{-2R_1} \tag{1.15}$$

$$D_2 \geq \sigma^2 2^{-2R_2} \tag{1.16}$$

and

$$D_0 \geq \begin{cases} \sigma^2 2^{-2(R_1+R_2)} & \text{if } D_1 + D_2 > \sigma^2 + D_0 \\ \frac{\sigma^2 2^{-2(R_1+R_2)}}{1-(\sqrt{(1-D_1)(1-D_2)}-\sqrt{D_1 D_2 - 2^{-2(R_1+R_2)}})^2} & \text{otherwise} \end{cases} \tag{1.17}$$

This is the only case for which the rate distortion region is completely known.

1.2 Joint Source-Channel Decoding of Multiple Description

As already mentioned, MDC is a source coding method which also involves systematic addition of redundancy to achieve reliability over erasure channels. In (n channel) MDQ, the addition of redundancy is mainly done in the IA by assigning only a fraction of the

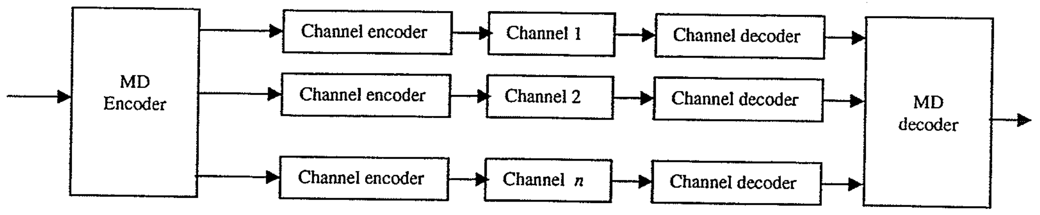


Figure 1.7: Separate decoding of multiple description.

possible index n-tuples. In case of MD correlating transforms, the redundancy is added by the linear transform which introduces an extra correlation. In any method of MD coding, the added redundancy exists in the form of inter-description correlation.

Although the main consideration in MDC is channel erasures, the transmission over practically available channels introduces random bit errors. Consider an n description MD coded system which uses error control coding to combat random bit errors. The standard way of channel coding in this situation is to encode the descriptions separately and to send them over separate channels as shown in the Figure 1.7. At the receiver they are decoded separately by respective decoders and passed to the MD decoder. Hereafter, we call this method of decoding as *separate decoding.*

It is a well known fact that when there is redundancy present in the source encoder output, it can be used to improve the error correcting capability of a channel code [12], [13]. Therefore, when more than one description is received, it would be possible to decode the channel codes of received descriptions jointly in a way that utilizes the inter description correlation constructively to improve the error correcting capability. Moreover, MDC can be considered as a joint source-channel code as explained previously. For example, we can view the IA in a MDQ as a sort of error detecting code since it does not use all the index n-tuples (in case of two channel MDQ, an invalid index pair at the channel output indicates an error) When this joint source-channel code is combined with an additional level of error correction coding, it would be possible to implement a decoder that uses the redundancy added by both MD coding and channel coding to correct the errors introduced in the channel. This problem, which we refer to as joint source-channel decoding of multiple description, is the main topic of this thesis. The objective of our work is to study this problem and to develop efficient joint decoding techniques that can be applied in a MDC environment.

This thesis is organized as follows.

- The previous related work is discussed in Chapter 2.
- Chapter 3 presents a list Viterbi algorithm based joint decoding scheme which is capable of achieving a significant performance improvement over separate decoding at a low complexity.
- Chapter 4 discusses bit level joint trellis decoding which is an extension to the optimum joint trellis decoding presented in [14].
- Chapter 5 focuses on the index reassignment problem in MDQ.
- Chapter 6 gives a summary of the thesis, some conclusions and possible future work.

Chapter 2

Previous Related Work

In this chapter, we discuss previous work in joint source-channel decoding of multiple descriptions. Joint decoding schemes (we use the term joint decoding to mean joint source-channel decoding of multiple descriptions) can be categorized into two types as iterative and non-iterative decoding. Sirinivasan et al. in [15] presented the first iterative type joint decoding scheme which works in a manner similar to turbo decoding [16]. Similar, but somewhat improved iterative decoding scheme can be found in [17] which uses the concept of "extrinsic information" for exchanging information between constituent decoders. Yahampath et al. in [14] presented two type of non-iterative decoding schemes which exploit the inter-description correlation explicitly in the Viterbi decoding algorithm [18].

Section 2.1 and 2.2 discuss the iterative and non-iterative joint decoding schemes respectively. An experimental evaluation of the schemes is given in Section 2.3 followed by a comparison in Section 2.4.

2.1 Iterative Type Joint Decoding

Iterative type joint decoding schemes borrow their concept from turbo coding. Therefore, a brief discussion of turbo coding would facilitate the understanding of the iterative joint decoding schemes.

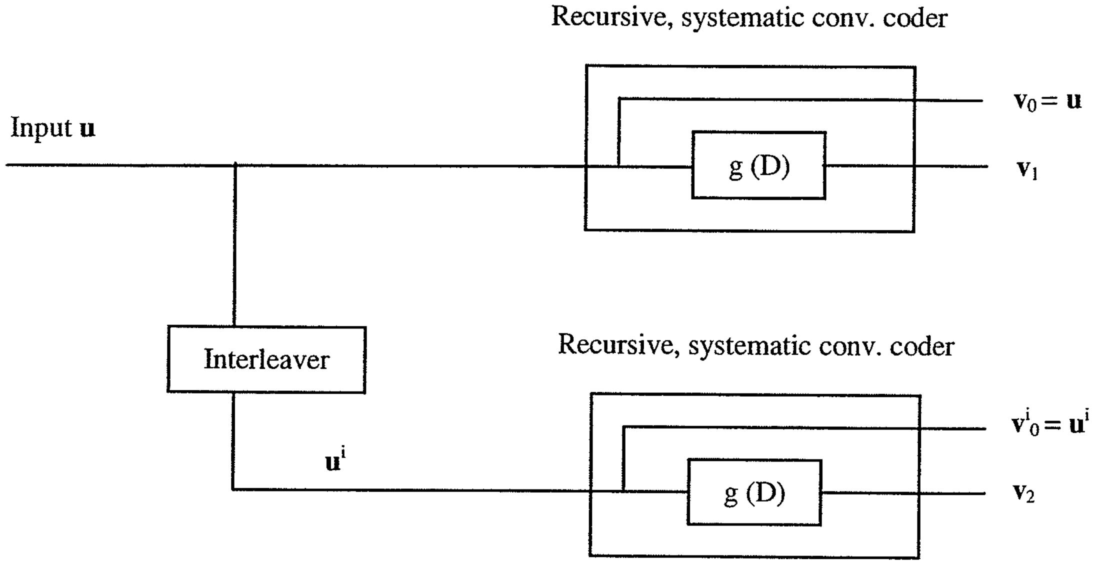

Recursive, systematic conv. coder
Input u
g (D)
$v_0 = u$
v_1
Interleaver
Recursive, systematic conv. coder
u^i
g (D)
$v^i_0 = u^i$
v_2

Figure 2.1: A turbo encoder.

Figure 2.1 shows a turbo encoder. A length L input bit sequence $\mathbf{u}$=$(u_1, u_2, \ldots, u_L)$ is encoded by two systematic feedback convolutional encoders (description of a systematic feedback convolutional encoder can be found in Section 11.1 of [19]). The input $\mathbf{u}$ is directly fed to the encoder 1 while an interleaved version $\mathbf{u}^i$ is fed to the encoder 2. Therefore, the systematic parts $\mathbf{v}_0$ and $\mathbf{v}_0^i$ (each equal to the input of the respective encoder) are just interleaved versions of each other and therefore, in general, only one of them (assume it to be $\mathbf{v}_0$ in this case) is transmitted. Figure 2.2 shows the turbo decoder for the encoder in Figure 2.1. Let $\mathbf{r}_0$, $\mathbf{r}_1$ and $\mathbf{r}_2$ be the channel outputs for $\mathbf{v}_0$, $\mathbf{v}_1$ and $\mathbf{v}_2$ respectively. The MAP decoders estimate the a posteriori log likelihood ratios of the input bits to the corresponding encoders by using a posteriori probability estimation algorithm such as the BCJR algorithm [20]. For example, the estimated a posteriori log likelihood ratio $L_1(u_t)$ of the input bit u_t (at discrete time t) in the MAP decoder 1 is given by

$$L_1(u_t) = \log \frac{P(u_t = 1|\mathbf{r}_0, \mathbf{r}_1)}{P(u_t = 0|\mathbf{r}_0, \mathbf{r}_1)}. \tag{2.1}$$

where $P(u_t = j|\mathbf{r}_0, \mathbf{r}_1)$, $j = 0, 1$, is the a posteriori probability of the event that the input

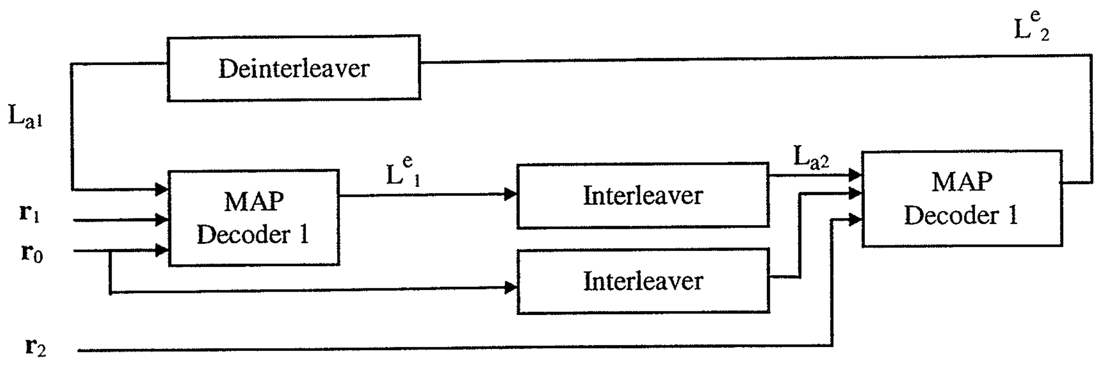


Figure 2.2: Turbo decoder for encoder in Figure 2.1.

bit u_t is equal to j given $\mathbf{r}_0$ and $\mathbf{r}_1$. This is also called as a posteriori L-value of u_t. The calculation of (2.1) requires the priori log likelihood ratio $L_{1a}(u_t)$, also called priori L-value of u_t, given by

$$L_{1a}(u_t) = \log \frac{P(u_t = 1)}{P(u_t = 0)}. \tag{2.2}$$

where $P(u_t = 1)$ is the priori probability of the event that the input bit u_t equals to 1. From the a posteriori L-value in (2.1), the effect of the current input bit is subtracted to obtain a quantity called *extrinsic information* denoted by $L^e_1(u_t)$. This represents the information about the bit u_t given by the other bits (due to parity constraints). Now, this extrinsic information (with suitable interleaving) is passed to the MAP decoder 2 as its priori information $L_{2a}(u^i_t)$. The extrinsic information obtained in the decoder 2 is, again, fed back to the encoder 1 as its priori information. This process is carried out iteratively and it improves the estimate of the input $\mathbf{u}$ in each decoder in successive iterations. However, the amount of improvement achieved in an iteration diminishes very rapidly and finally it comes to a point where no further improvement is possible. Therefore, the above procedure is iterated only a limited number of times, i.e. until the improvement gained in one more iteration is not worth the extra computational complexity needed. At this point, the encoder 2 input

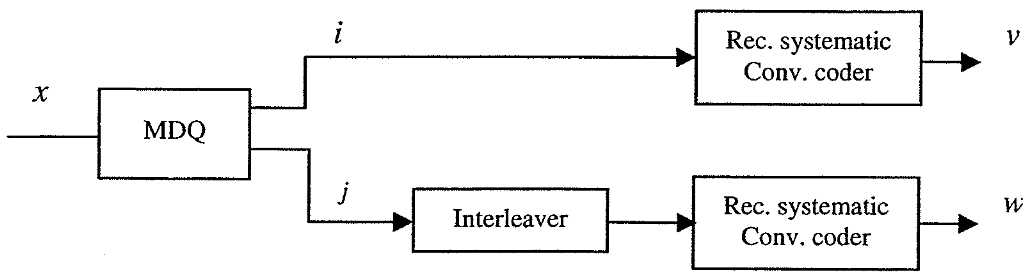


Figure 2.3: Channel coding of MDQ output for iterative decoding.

is estimated by the decoder 2 using

$$\hat{u}_t^i = \begin{cases} 1 & \text{if } L_2(u_t^i) \geq 0 \\ 0 & \text{otherwise.} \end{cases} \tag{2.3}$$

This estimated bit sequence is deinterleaved to obtain the estimate $\hat{\mathbf{u}}$ of the turbo encoder input. Therefore, the basic idea behind turbo coding can be summarized as follows.

- Two correlated sequences are channel coded separately and transmitted.
- At the receiver, decoder 1 extracts the information about the input to the encoder 1 which is then used to derive information about the input to the encoder 2 (using the correlation). This information is then passed as the priori information to the decoder 2 which in turn derives and pass the priori information to the decoder 1. This process is carried out iteratively a pre-determined number of times.

We know that the descriptions generated by an MDC are correlated and therefore, when those are separately channel coded before transmission, it would be possible to use an iterative decoding scheme as in turbo decoding at the receiver. We first discuss the iterative decoding scheme presented in [15]. Figures 2.3 and 2.4 show a 2 channel MDQ system that uses iterative decoding. Assume that $R_1 = R_2 = r$ bits/description. Let the source sequence $x_1^L = (x_1, x_2, \ldots, x_L)$ be MD encoded and let (i_1^L, j_1^L) be the corresponding index sequences. The sequences i_1^L and an interleaved version of j_1^L are then encoded by systematic feedback convolutional encoder to obtain the channel input sequence v_1^L and w_1^L. Note that, unlike

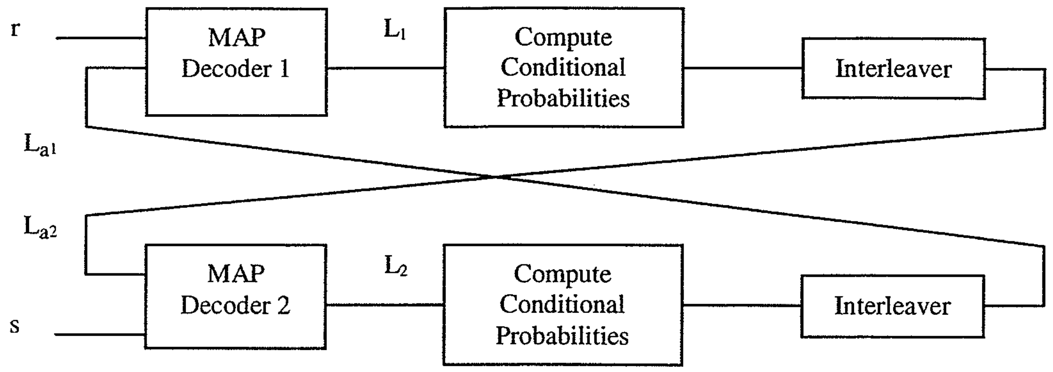

r
MAP Decoder 1
L1
Compute Conditional Probabilities
Interleaver
La1
La2
MAP Decoder 2
L2
Compute Conditional Probabilities
Interleaver
s

Figure 2.4: Iterative decoder for encoding in Figure 2.3.

in turbo coding, the systematic parts of the channel coder outputs in this case are not interleaved versions of each other. Let $(i_{k1}, i_{k2}, \ldots, i_{kr})$ be the binary representation of the index i_k at time k. Consider the case where both r_1^L and s_1^L are received. From these received sequences, the MAP decoders estimate the a posteriori log likelihood ratios (a posteriori L-values). For example, the a posteriori L-value $L_1(i_{kn})$ of the bit i_{kn}, which is estimated by the MAP decoder 1, is given by

$$L_1(i_{kn}) = \log \frac{P(i_{kn} = 1|v_1^L)}{P(i_{kn} = 0|v_1^L)}. \tag{2.4}$$

The calculation of the above a posteriori L-value requires the priori L-value $L_{a1}(i_{kn})$ of the bit i_{kn}. It is straight froward to calculate the a posteriori probability distribution $P(i_{kn}|v_1^L)$ from the a posteriori L-value calculated above. As an example,

$$P(i_{kn} = 1|v_1^L) = \frac{e^{L_1(i_{kn})}}{1 + e^{L_1(i_{kn})}}.$$

Now, the a posteriori probability distribution of the description 1 at time k is calculated using the relation

$$P(i_k = a|v_1^L) = \prod_{n=1}^{r} P(i_{kn} = a_n|v_1^L), \qquad a \in \mathcal{I} = \{0, 1, \ldots, 2^r - 1\} \tag{2.5}$$

where $(a_1, a_2, \ldots, a_r)$ is the binary representation of a. From the probability distribution in (2.5) it is possible to calculate the a posteriori probability distribution of the description 2 at time k by using the relation

$$P(j_k = b) = \sum_{a=0}^{2^R-1} P(j_k = b | i_k = a) P(i_k = a) \qquad b \in \mathcal{J} = \{0, 1, \ldots, 2^r - 1\}. \tag{2.6}$$

The conditional probability $P(j_k = b | i_k = a)$ in (2.6) mainly depends on the IA. Let $(b_1, b_2, \ldots, b_r)$ be the binary representation of the index b. From the probability distribution of the description 2 at time k, the probability distribution of the bit j_{kn} is found as follows

$$P(j_{kn} = 1) = \sum_{b:b_n=1} P(j_k = b)$$

and

$$P(j_{kn} = 0) = 1 - P(j_{kn} = 1).$$

Now, these probabilities are used as priori information for the MAP decoder 2 to calculate the priori L-value $L_{a2}(j_{kn})$ as

$$L_{a2}(j_{kn}) = \log \frac{P(j_{kn} = 1)}{P(j_{kn} = 0)}. \tag{2.7}$$

In the same way, the a posteriori L-values estimated by the MAP decoder 2 are used to obtain the priori L-values for the MAP decoder 1 and this process is iterated several times as in turbo decoding. After a fixed number of iterations the bits i_{kn} and j_{kn} are estimated using a posteriori L-values of the MAP decoders. For example, the bit i_{kn} is estimated as

$$\hat{i}_{kn} = \begin{cases} 1 & \text{if } L_1(i_{kn}) \geq 0 \\ 0 & \text{otherwise.} \end{cases} \tag{2.8}$$

A careful analysis shows that the iterative decoding in [17] is very similar to that in [15]. The main difference is that [17] uses extrinsic information ($L_1^e(i_{kn})$ and $L_2^e(j_{kn})$) to calculate

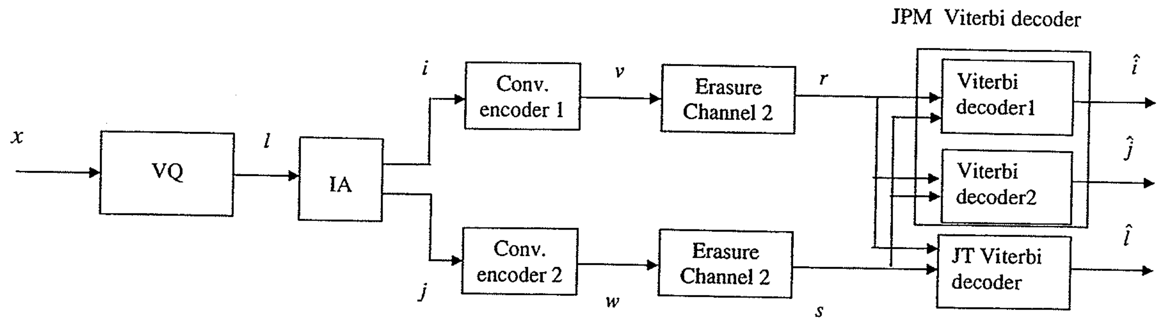


Figure 2.5: Non-iterative joint decoding.

the priori log likelihood ratios $(L_{a2}(j_{kn})$ and $(L_{a1}(i_{kn}))$ while, as seen above, the iterative decoding in [15] uses the a posteriori L-values directly to calculate the priori information. The extrinsic information in [17] is calculated as follows.

$$L_1^e(i_{kn}) = L_1(i_{kn}) - (L_{a1}(i_{kn}))_{prev} \tag{2.9}$$

Here, $(L_{a1}(i_{kn}))_{prev}$ is the priori L-value estimated in the previous iteration using the extrinsic information from the MAP decoder 2. i.e it is the priori information used in estimating $L_1(i_{kn})$ in (2.9). Similarly

$$L_2^e(j_{kn}) = L_2(j_{kn}) - (L_{a2}(j_{kn}))_{prev}. \tag{2.10}$$

2.2 Non-Iterative Joint Decoding

The iterative decoding algorithms discussed above require an interleaver which requires modification to the encoder. On the other hand, non-iterative joint decoding algorithms exploit the redundancy introduced by the MD coder without requiring any change at the encoder side. Two methods of non-iterative joint decoding schemes, namely *joint trellis decoding* and *joint path metric decoding*, are given in [14]. Figure 2.5 shows both joint trellis

(JT) decoder and joint path metric (JPM) decoder in a single diagram. Let's assume that a two channel MDQ is used and each description is convolutional encoded separately by a convolutional encoder where there is a one to one correspondence between the convolutional encoder input and the MDQ output alphabets (i.e. if the MDQ has rates $R_1 = r_1$ and $R_2 = r_2$ bits/description, then the convolutional encoders need to have r_1 and r_2 bit inputs respectively). A generalization to the joint trellis decoding is developed in Chapter 4, which does not require this one-to-one correspondence. Let $l_1^L = (l_1, l_2, \ldots, l_L)$ be the index sequence for the source sequence $\mathbf{x}_1^L = (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_L)$ and let (i_1^L, j_1^L), (v_1^L, w_1^L) and (r_1^L, s_1^L) be corresponding description, channel input, and channel output sequences respectively.

Joint Trellis Decoding

In this method, the two convolutional encoders are viewed as a single super convolutional encoder with the input (i_k, j_k) and the output (v_k, w_k). For this super convolutional coder, a super-trellis (or joint trellis) is built. A state S^J of the joint trellis is given by the pair of states (S_1, S_2) of the individual encoders. The Viterbi algorithm (see Appendix A) is used to search this trellis to find the optimum sequence estimate (i_1^{*L}, j_1^{*L}). We know that an IA is a one-to-one mapping and hence the index pair (i_k, j_k) is equivalent to the index l_k. Therefore, the MAP estimate of the sequence l_1^L is given by

$$\begin{aligned} l_1^{*L} &= \arg\max_{l_1^L} P(l_1^L | r_1^L, s_1^L) \\ &= \arg\max_{(i_1^L, j_1^L)} P(i_1^L, j_1^L | r_1^L, s_1^L) && (2.11) \\ &= \arg\max_{(i_1^L, j_1^L)} \frac{P(r_1^L, s_1^L, i_1^L, j_1^L)}{P(r_1^L, s_1^L)} && (2.12) \\ &= \arg\max_{(i_1^L, j_1^L)} P(r_1^L, s_1^L, i_1^L, j_1^L) && (2.13) \end{aligned}$$

The simplification from (2.12) to (2.13) is due to the fact that the term $P(r_1^L, s_1^L)$ is constant given the particular channel output and does not affect the maximization in (2.12). We call the above MAP sequence estimate as the JT-MAP estimate. Our objective here is to find the MAP sequence estimate given in (2.13) by using the Viterbi algorithm. For that, as explained in Appendix A, we would have to use a monotonic function of $P(i_1^L, j_1^L, r_1^L, s_1^L)$ as the *path metric* which can be expanded into a summation of terms called *branch metrics* that corresponds to individual branches of a path along the trellis. This can be achieved first by expressing the probability $P(i_1^L, j_1^L, r_1^L, s_1^L)$ as a multiplication of probability terms each related to a separate branch in the path and then taking the logarithm. We see that

$$\begin{aligned} P(r_1^L, s_1^L, i_1^L, j_1^L) &= P(r_1^L, s_1^L | i_1^L, j_1^L)\ P(i_1^L, j_1^L) \\ &= P(r_1^L | i_1^L) P(s_1^L | j_1^L)\ P(i_1^L, j_1^L) && (2.14) \\ &= P(r_1^L | v_1^L) P(s_1^L | w_1^L)\ P(i_1^L, j_1^L) && (2.15) \\ &= \prod_{k=1}^{L} P(r_k | v_k) P(s_k | w_k) P(i_1^L, j_1^L) && (2.16) \\ \therefore \quad \log P(i_1^L, j_1^L | r_1^L, s_1^L) &= \sum_{k=1}^{L} \log P(r_k | v_k) + \log P(s_k | w_k) + \log P(i_1^L, j_1^L) && (2.17) \end{aligned}$$

The step from (2.14) to (2.15) is possible since the input sequences i_1^L and j_1^L uniquely determine the encoder outputs v_1^L and w_1^L respectively. The simplification in (2.16) is due to the memoryless nature of the channel. To simplify $P(i_1^L, j_1^L)$ further, we have to assume a model for the source. For example, if the process $\{x_k\}$ is i.i.d., then the process $\{(i_k, j_k)\}$ is also i.i.d. and the term $\log P(i_1^L, j_1^L)$ in (2.17) can be expanded as

$$\begin{aligned} \log P(i_1^L, j_1^L) &= \log \prod_{k=1}^{L} P(i_k, j_k) \\ &= \sum_{k=1}^{L} \log P(i_k, j_k). && (2.18) \end{aligned}$$

and therefore the branch metric $(bm_k)_{i.i.d-JT-MAP}$ used in the Viterbi algorithm in this case is

$$(bm_k)_{i.i.d-JT-MAP} = \log P(r_k | v_k) + \log P(s_k | w_k) + \log P(i_k, j_k).$$

On the other hand, if the source sequence is correlated, a better model would be a Markov process. If the process $\{\mathbf{x}_k\}$ is first-order Markov, then

$$P(\mathbf{x}_k|\mathbf{x}_{k-1},\mathbf{x}_{k-2},\mathbf{x}_{k-3}\ldots) = P(\mathbf{x}_k|\mathbf{x}_{k-1}) \tag{2.19}$$

and

$$P(\mathbf{x}_1^L) = P(\mathbf{x}_1)\prod_{k=2}^{L} P(\mathbf{x}_k|\mathbf{x}_{k-1}). \tag{2.20}$$

In this case, it is common to approximate the process $\{l_k\}$ also to be Markov. Therefore

$$\log P(i_1^L, j_1^L) = \sum_{k=1}^{L} \log P(i_k, j_k|i_{k-1}, j_{k-1})$$

where

$$P(i_k, j_k|i_{k-1}, j_{k-1}) = P(i_1, j_1) \qquad \text{when } k = 1$$

and the corresponding branch metric is given by

$$(bm_k)_{M-JT-MAP} = \log P(r_k|v_k) + \log P(s_k|w_k) + \log P(i_k, j_k|i_{k-1}, j_{k-1}).$$

The ML estimate of the sequence l_1^L is given by

$$\hat{l}_1^L = \arg\max_{l_1^L}\ P(r_1^L, s_1^L|l_1^L).$$

But

$$\begin{aligned} P(r_1^L, s_1^L | l_1^L) &= P(r_1^L, s_1^L | i_1^L, j_1^L) \\ &= P(r_1^L | i_1^L) P(s_1^L | j_1^L) \\ &= P(r_1^L | v_1^L) P(s_1^L | w_1^L) \\ &= \prod_{k=1}^{L} P(r_k | v_k) P(s_k | w_k) \end{aligned} \tag{2.21}$$

Now, let's take $\log P(r_1^L, s_1^L | i_1^L, j_1^L)$ as our path metric. Therefore, for any source model, the branch metric used in the Viterbi algorithm is given by

$$(bm_k)_{JT-ML} = \log P(r_k | v_k) + \log P(s_k | w_k).$$

When this branch metric is used in the Viterbi algorithm to search the joint trellis, the resulting scheme is called as the JT-ML algorithm.

Joint Path Metric decoding

In joint path metric decoding, a separate decoder is used in each channel. However, in order to exploit the inter-description redundancy, the full channel output (r_1^L, s_1^L) is used in each decoder in Viterbi sequence estimation. For example, the MAP estimate of the sequence i_1^L given (r_1^L, s_1^L) is obtained by

$$i_1^{*L} = \arg\max_{i_1^L} \; P(i_1^L | r_1^L, s_1^L)$$

which can be expanded as

$$\begin{aligned}\arg\max_{i_1^L} P(i_1^L|r_1^L,s_1^L) &= \arg\max_{i_1^L} P(i_1^L,r_1^L,s_1^L) \\ &= \arg\max_{i_1^L} P(r_1^L,s_1^L|i_1^L)P(i_1^L) \\ &= \arg\max_{i_1^L} \prod_{k=1}^{L} P(r_k,s_k|i_k)\ P(i_1^L) \qquad (2.22)\end{aligned}$$

We will refer to this as joint path metric MAP (JPM-MAP) estimate. In the case of an i.i.d. source, the term $P(i_1^L)$ can be expanded as

$$P(i_1^L) = \prod_{k=1}^{L} P(i_k)$$

and therefore, taking $\log P(i_1^L|r_1^L,s_1^L)$ as the path metric, we get the branch metric

$$(bm_k)_{i.i.d-JPM-MAP} = \log P(r_k,s_k|i_k) + \log P(i_k).$$

If the source is correlated $\{i_k\}$ can be approximated as first-order Markov. Then

$$P(i_1^L) = \prod_{k=1}^{L} P(i_k|i_{k-1})$$

where

$$P(i_k|i_{k-1}) = P(i_1) \qquad \text{when } k=1.$$

Therefore, the branch metric in this case is given by

$$(bm_k)_{M-JPM-MAP} = \log P(r_k,s_k|i_k) + \log P(i_k|i_{k-1}).$$

Given (r_1^L,s_1^L), the ML sequence estimate of i_1^L is obtained by

$$\hat{i}_1^L = \arg\max_{i_1^L} P(r_1^L,s_1^L|i_1^L).$$

We will refer to this as joint path metric ML (JPM-ML) estimate. It can be shown that for any source model, the branch metric for this estimation is given by

$$(bm_k)_{JPM-ML} = \log P(r_k, s_k | i_k).$$

Separate Decoding

In this chapter and in succeeding chapters, the performance of joint decoding is compared with separate decoding. In *separate MAP algorithm*, codewords in individual channels are separately decoded with a Viterbi decoder to obtain the MAP estimates of the corresponding index sequence. For example, the separate MAP estimate of the index sequence i_1^L is given by

$$i_1^{*L} = \arg\max_{i_1^L} P(i_1^L | r_1^L)$$

which can be found by using the Viterbi algorithm as explained in Appendix A. Similarly, *separate ML algorithm* finds the ML sequence estimate of the index sequence in each channel.

2.3 Simulation Results

Figures 2.6 and 2.7 are the experimental results presented in [14]. The non-iterative decoding schemes presented in [14] and the iterative decoding scheme in [15] are experimentally evaluated for an i.i.d. gaussian source and a Gauss Markov (GM) source which is given by

$$x_t = 0.9x_{t-1} + w_t$$

where $\{w_t\}$ is a unit variance Gaussian random process and t denotes discrete time. The above GM source is a simple model for practical sources such as speech and image samples. Source samples are MD coded at rate $R_1 = R_2 = 3$ bits/description. MDSQ is designed as described in chapter 1 with $\mu_0 = 1$ and $\mu_1 = \mu_2 = 0.005$. The index assignment is given

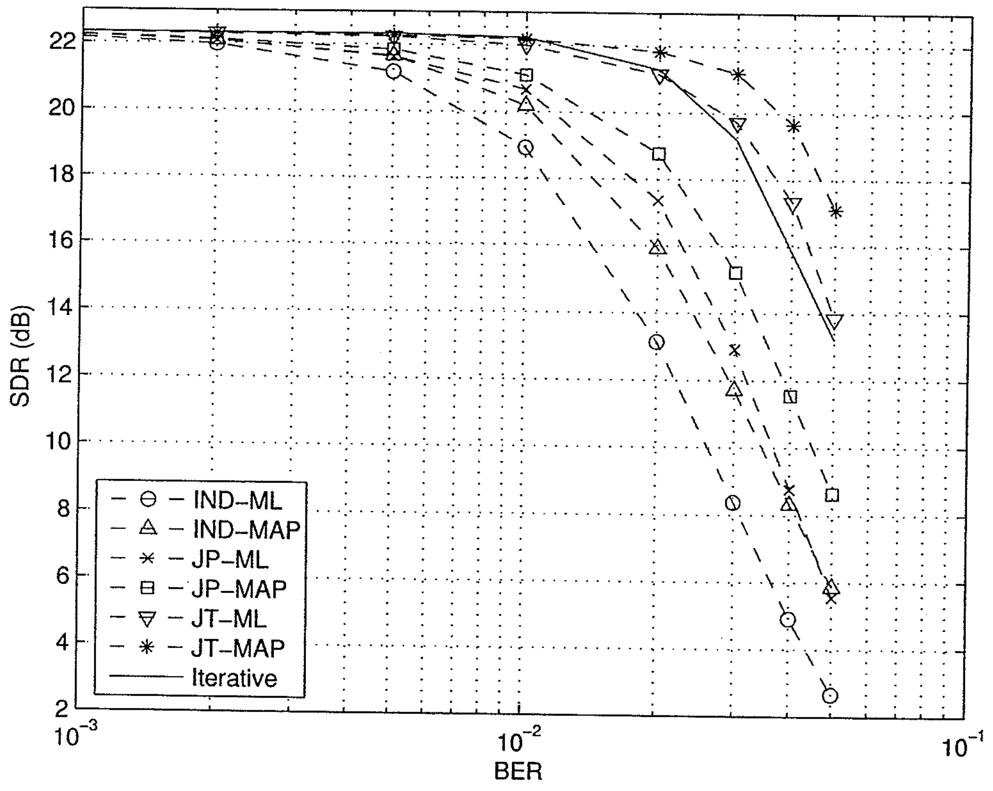


Figure 2.6: Performance variation of joint decoding and separate decoding with channel bit error probability for an i.i.d. Gaussian source of unit variance. Curves marked with IND-MAP and IND-ML denote separate MAP and separate ML respectively. The curves marked with JP-MAP and JP-ML denote JPM-MAP and JPM-ML respectively.

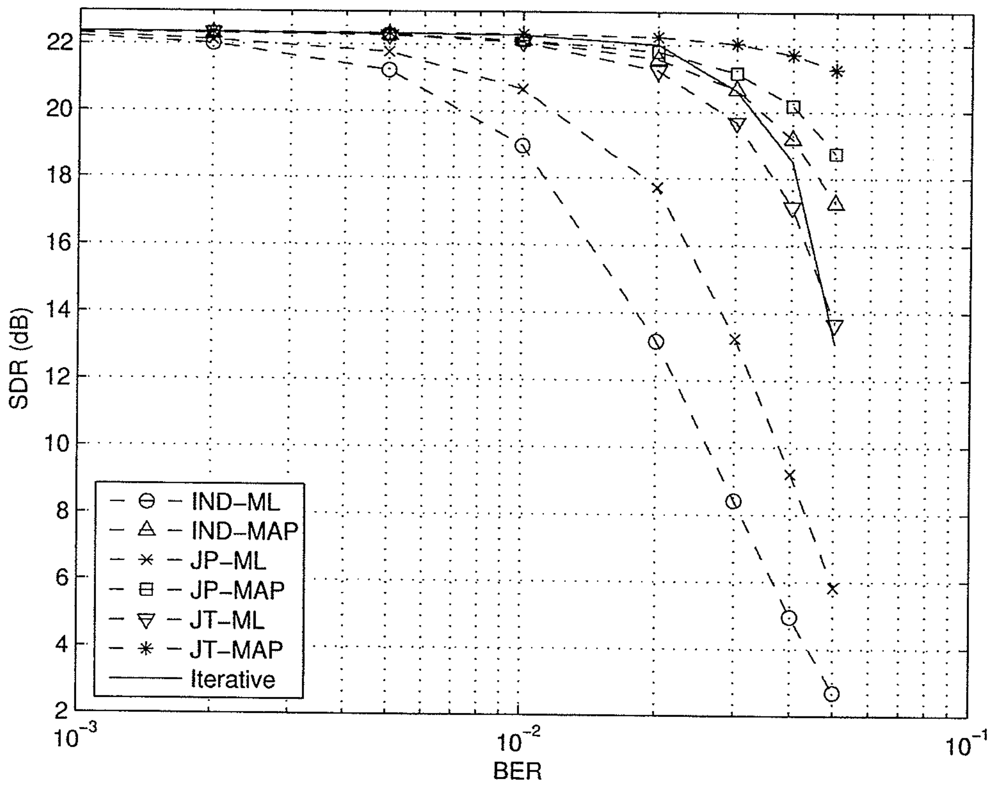


Figure 2.7: Performance variation of joint decoding and separate decoding with channel bit error probability for a Gauss Markov source. Curves marked with IND-MAP and IND-ML denote separate MAP and separate ML respectively. The curves marked with JP-MAP and JP-ML denote JPM-MAP and JPM-ML respectively.

in Figure 2.8. For the evaluation of non-iterative joint decoding schemes, descriptions are convolutional coded with a (4,3,2) (i.e. 4 outputs, 3 inputs and memory 2) code of constraint length 6, which is obtained from Table V of [21]. A block of 128 MDSQ output indices (of a description) is convolutional coded to obtain the 512 bit long channel codewords. For the system with iterative decoding, rate 3/4 convolutional encoders are obtained by puncturing a rate 1/2 binary code with a constraint length of 6. An interleaver of length 1024 bits is used on one of the MD channels prior to channel coding. Thus, from the channel coding point of view, this system is comparable to the other systems considered here (except the delay caused by the interleaver). We use as the figure of merit, the average signal to distortion ratio

$$SDR = EX^2/E(X-\hat{X})^2$$

where $\hat{X}$ is the reconstruction of X at the decoder. To find the improvement achieved by joint decoding, the performance of separate decoding is also evaluated (with both MAP and ML sequence estimations). With all of the decoding algorithms, the mean value of the source is selected as the output when the decoded index pair is invalid. The source statistics required in the decoding schemes are empirically estimated from a training sequence. We assume a binary symmetric channel (BSC) model in the experiment. Figures 2.6 and 2.7 show the variation of the performance of each decoding scheme with the channel bit error probability for an i.i.d. source and a GM source respectively. It can be seen that the joint decoding schemes achieve a significant improvement over separate decoding. The results suggest that JT and iterative decoding schemes are very efficient in exploiting the redundancy present in multiple descriptions. JPM decoding algorithms have significantly lower performance than JT and iterative decoding in the case of an i.i.d. source. Therefore, it can be concluded that JPM decoding algorithms are less efficient in exploiting the redundancy in multiple descriptions. The JT-MAP algorithm has the best performance out of all the joint decoding algorithms considered. It is observed that MAP decoding algorithms always perform better than their ML counterparts. For an i.i.d. source this is due to the

0	1						
2	3	5					
	4	6	7				
		8	9	10			
				11	13		
				12	14	15	
					16	17	19
						18	20

Figure 2.8: The IA matrix used in the experimental evaluation of joint decoding schemes.

redundancy caused by the non-uniformity in the MDQ output statistics. The performance difference between MAP and ML decoding is considerably higher in the case of GM source, since there is a large amount of redundancy in the source sequence in the form of temporal correlation.

2.4 Comparison of Joint Decoding Schemes

In this section we compare the joint decoding schemes presented in this chapter. The main considerations in the comparison are

- efficiency in exploiting the redundancy present in multiple descriptions
- computational complexity
- decoding delay

The iterative decoding schemes are highly efficient in exploiting the redundancy. However, they posses some disadvantages. For example, several iterations are required to decode each bit and this can result in a high decoding delay. Furthermore, to obtain a good performance, long interleavers must be employed and the longer the interleaver, the higher the decoding delay. As explained in Chapter 1, one of the main application for MDC is in real time communication systems, where the allowable delay is restricted. In such systems, the application of the iterative decoding may be restricted due to their longer delays. Another disadvantage of using interleavers is that one has to modify the encoder.

Joint trellis decoding is an optimum sequence estimation algorithm and therefore, it is very efficient in utilizing the redundancy present in descriptions to improve the performance. With this algorithm, channel codewords can be decoded as they arrive, and therefore, there is no increase in the decoding delay compared to separate decoding of the same code. The main disadvantage in the joint trellis decoding is the increased computational complexity. If the convolutional encoders have constraint lengths ν_1 and ν_2 respectively there can be $2^{\nu_1+\nu_2}$ states in the joint trellis whereas the separate trellises have only 2^{ν_1} and 2^{ν_2} states respectively. This means that the number of states in joint trellis is the multiplication of the number of states in separate trellises and therefore, one can expect a similar increase in the computational complexity. However, the actual number of states in the joint trellis is typically less than $2^{\nu_1+\nu_2}$ since not all the index pairs (i, j) are necessarily used in the IA matrix. To make the analysis simple, assume that $\nu_1 = R_1 m_1$ and $\nu_2 = R_2 m_2$ where R_1, R_2 are the number of inputs (which are also the rates of MDQ in bits/description) and m_1, m_2 are the memory of the respective convolutional coders. In one extreme, when only one diagonal is used in the IA matrix, the two descriptions will be identical and therefore a state of the super convolutional coder is equal to the past n MDQ output indices of one description where $n = \max(m_1, m_2)$. In this case, there will be $\max(2^{\nu_1}, 2^{\nu_2})$ states in the joint trellis. On the other extreme, when all the index pairs are used in the IA matrix, the number of states in the joint trellis would be $2^{\nu_1+\nu_2}$.

Another disadvantage of the joint trellis decoding scheme is that its complexity increases exponentially with the rates R_1 and R_2. Each state of the joint trellis has $(2^{R_1+R_2})c$ arriving paths, where c is the fraction of the index pairs used. If we assume c to be a constant, the number of additions and (two-way) comparisons required in the Viterbi algorithm increases exponentially with R_1 and R_2. On the other hand, it would be impossible to have one-to-one correspondence between the convolutional coder input and MDQ output alphabets for large description alphabets (description alphabet sizes in turn depend on the rates R_1 and R_2), since it results in a large number of states in the trellises. In Chapter 4, we introduce a bit level joint trellis decoding scheme which eliminates the requirement of one-to-one correspondence between MDQ output alphabets and convolutional coder input alphabets. It also eliminates the above mentioned exponential increase in the computational complexity.

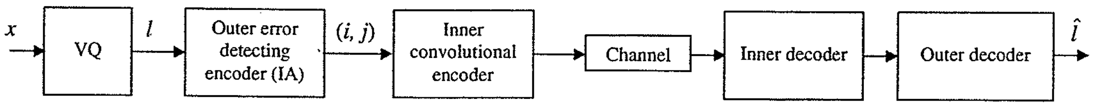
Finally, the JPM decoding is only slightly more complex than separate decoding and the decoding delay is the same. However, it is suboptimal and therefore less efficient than iterative decoding and joint trellis decoding in exploiting the redundancy in multiple descriptions.

Chapter 3

List Viterbi Algorithm Based Joint Decoding

Multiple description joint decoding techniques such as iterative decoding and joint trellis decoding are very efficient in exploiting the redundancy introduced in the process of multiple description encoding. But, these schemes possess some drawbacks which restrict the applicability in certain situations as pointed out in Chapter 2. On the other hand, the joint path metric decoding is only little more complex than separate decoding and would have less restrictions in the applicability. But at the same time it is significantly less efficient than iterative decoding and joint trellis decoding. Therefore, there exists the challenge of finding less complex and fairly efficient techniques which can be used in applications where other efficient schemes cannot be used. In this chapter, we study a list Viterbi algorithm [22] based joint decoding scheme which can be combined with joint path metric decoding methods (described in Chapter 2) to enhance their performance [23].

A multiple description code generated by MDQ can be considered as a nonlinear error detection code of rate $\log(Q)/2r$ [17] where r is the number of bits per description and Q is the number of elements that are used in the IA matrix. As explained in Chapter 1, the index assignment is analogous to channel coding since it uses only Q pairs out of 2^{2r} all possible index pairs. Therefore, the channel coding of multiple descriptions for the transmission



x
VQ
l
Outer error detecting encoder (IA)
(i, j)
Inner convolutional encoder
Channel
Inner decoder
Outer decoder
$\hat{l}$

Figure 3.1: Concatenated coding based joint decoding of multiple descriptions

over a noisy channel can be considered as a sort of concatenated coding. As an example, convolutional encoding of multiple descriptions can be viewed as a concatenated coding with a error correcting inner convolutional code and a error detecting outer multiple description block code. Therefore, following the same analogy, it would be possible to device some concatenated decoding algorithm for jointly decoding the multiple descriptions as shown in Fig 3.1. Concatenated coding has been widely used in digital communication and well studied in coding theory. The inner code is usually decoded with soft decision decoding and the outer code is decoded with hard decision decoding.

There are many decoding algorithms such as BCJR algorithm [20], soft output Viterbi algorithm (SOVA) [24] and list Viterbi algorithm (LVA) [22] etc. that can be used in decoding the inner convolutional code. The BCJR algorithm is an optimum decoding algorithm that gives the a posteriori probability of the source symbols. However, it requires a whole codeword be received before decoding and therefore is not suitable for continuous mode transmission. Various versions of the BCJR algorithm such as sliding window BCJR algorithm has been found to facilitate continuous mode transmission at the expense of optimality. In SOVA a reliability indicator is given for each decoded bit which can be used as a soft output in the subsequent decoding steps. However, this reliability indicator is not in general equal to the a posteriori probability and hence the algorithm is suboptimal. SOVA is a sequence estimation scheme whereas BCJR algorithm is a symbol by symbol estimation algorithm. SOVA can be used in continuous mode transmission just like the Viterbi algorithm. The LVA, on the other hand, is not a true soft decoding scheme since it does not give a soft output for each symbol separately. Instead, it finds a list of best estimates of the symbol sequence (and the reliability of each estimate if needed). There are several versions of the LVA for different applications and complexity requirements. But in all those cases the main factor determining the complexity is the list size. For list size one, the algorithm is essentially the Viterbi algorithm. Different versions of LVA can be found in [22], [25] and [26] where the main usage illustrated is in concatenated coding with inner error correction convolutional code and outer error detection code.

3.1 The List Viterbi Algorithm

The LVA is a generalization of the Viterbi algorithm to find a rank ordered list of L globally best sequence estimates. There are basically two types of LVA: (1) Parallel LVA, (2) Serial LVA. The parallel LVA finds L best sequence estimates simultaneously in a single trellis search. At each decoding step, in the Viterbi algorithm the lowest cost path arriving at each state is selected as a survivor and the final survivor (in case of a terminated code) is the globally best path. In the parallel LVA, instead of keeping one survivor for each state, an ordered list of L best paths arriving at the state are retained. It is clear that this gives the L globally best paths in a terminated code. The computational complexity of the parallel LVA is approximately L times that of the Viterbi algorithm. In the serial LVA, L best paths are found iteratively such that the k-th best path is found by using the knowledge of previously found $k-1$ paths. The average computational complexity of the serial LVA is less than that of the parallel LVA since it is not required to find the best L survivors for each state in each decoding step. However, the algorithm requires the whole sequence before finding the list of sequence estimates and hence is not suitable for continuous mode transmission. More details of the serial LVA can be found in [22]. When LVA is used in decoding a concatenated code with an inner convolutional code and an outer error detecting code, the best L candidate sequences are found by decoding the inner code and then testing for the outer code. The best candidate that satisfies the outer code is selected as the final output. If there exist no sequence that satisfies the outer code in the list, an error is declared.

In our system, we use the LVA to decode the inner code since it is well suited to the scenario considered, i.e. we have a concatenated code with an inner convolutional code and an outer error detection block code. Our objective is to exploit this outer error detection code for obtaining a better estimate for the source sequence. LVA is also simpler compared to other algorithms such as BCJR algorithm and SOVA for small list sizes and it is possible to have a performance-complexity compromise by changing the list size.

3.2 LVA Based Joint Decoder

Figure 3.2 shows a system which uses the LVA in joint decoding the multiple descriptions. Let $l_1^L = (l_1, l_2, \ldots, l_L)$ be the index sequence for the source sequence $x_1^L = (x_1, x_2, \ldots, x_L)$ and let (i_1^L, j_1^L), (v_1^L, w_1^L) and (r_1^L, s_1^L) be corresponding description, channel input, and channel output sequences respectively. Then, the MAP estimate of the sequence l_1^L is given by

$$l_1^{*L} = \arg\max_{l_1^L} P(l_1^L | r_1^L, s_1^L).$$

From (2.11), (2.12) and (2.13)

$$\arg\max_{l_1^L} P(l_1^L | r_1^L, s_1^L) = \arg\max_{(i_1^L, j_1^L)} P(r_1^L, s_1^L, i_1^L, j_1^L) \tag{3.1}$$

and

$$\begin{aligned} P(r_1^L, s_1^L, i_1^L, j_1^L) &= P(r_1^L, s_1^L | i_1^L, j_1^L)\, P(i_1^L, j_1^L) \\ &= P(r_1^L | i_1^L)\, P(s_1^L | j_1^L)\, P(i_1^L, j_1^L) \qquad (3.2) \\ &= P(r_1^L)\, P(s_1^L)\, P(i_1^L | r_1^L)\, P(j_1^L | s_1^L)\, \frac{P(i_1^L, j_1^L)}{P(i_1^L)\, P(j_1^L)} \\ \therefore \quad \arg\max_{(i_1^L, j_1^L)} P(i_1^L, j_1^L | r_1^L, s_1^L) &= \arg\max_{(i_1^L, j_1^L)} P(i_1^L | r_1^L)\, P(j_1^L | s_1^L)\, \frac{P(i_1^L, j_1^L)}{P(i_1^L)\, P(j_1^L)} \\ &= \arg\max_{(i_1^L, j_1^L)} \log P(i_1^L | r_1^L) + \log P(j_1^L | s_1^L) \\ &\quad + \log P(i_1^L, j_1^L) - \log P(i_1^L) - \log P(j_1^L) \qquad (3.3) \end{aligned}$$

Note that the term $P(i_1^L, j_1^L)/(P(i_1^L)\, P(j_1^L))$ takes into account the inter-description correlation while the terms $P(i_1^L | r_1^L)$ and $P(j_1^L | s_1^L)$ take into account the channel coding and the temporal correlation in the descriptions.

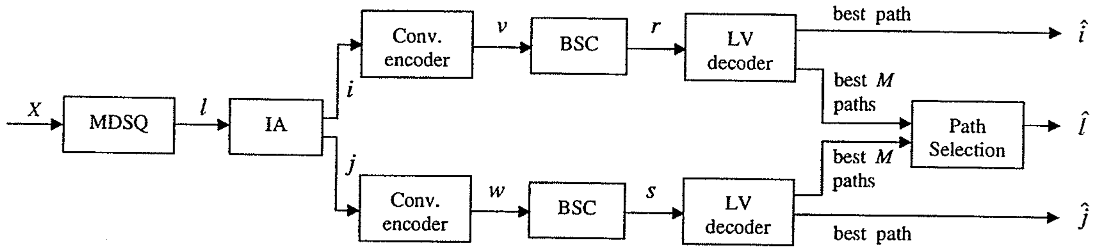


Figure 3.2: List Viterbi algorithm based multiple descriptions joint decoder

The brute force method of finding the MAP estimate of the sequence pair (i_1^L, j_1^L) would be to enumerate all the possible sequences i_1^L and j_1^L and then to apply (3.3). However, this is not practically feasible even for a moderate value of block length L, since 2^{2rL} sequence pairs have to be considered, where r is the number of bits per description. On the other hand, many of those pairs would have a much smaller a posteriori probability compared to the MAP sequence pair given by (3.3). Therefore, instead of considering all the possible sequences, we propose to select a limited number of potentially good candidates for i_1^L and j_1^L by using the LVA. The LVA based joint decoder works as follows:

- List Viterbi decoders 1 and 2 use the path metrics $\log P(i_1^L|r_1^L)$ and $\log P(j_1^L|s_1^L)$ to find M best estimates for i_1^L and j_1^L respectively. From Appendix A, for an i.id source, the corresponding branch metrics bm_{1k} and bm_{2k} of respective decoders are given by

$$\begin{aligned} bm_{1k} &= \log P(r_k|v_k) + \log P(i_k) \\ bm_{2k} &= \log P(s_k|w_k) + \log P(j_k) \end{aligned}$$

  where k denotes the time.

- From the resulting $M \times M$ sequence pairs (i_1^L, j_1^L) the one that minimizes the cost function

$$C = -\{\log P(i_1^L|r_1^L) + \log P(j_1^L|s_1^L) + \log P(i_1^L, j_1^L) - \log(P(i_1^L) - \log P(j_1^L)\}$$

is selected by the path selection unit as the output. The log probabilities $\log P(i_1^L|r_1^L)$ and $\log P(j_1^L|s_1^L)$ are given by the preceding list decoding stage. For an i.i.d. source, the remaining term is calculated by using the equation

$$\log P(i_1^L, j_1^L) - \log(P(i_1^L) - \log P(j_1^L) = \sum_{k=1}^{L} \log P(i_k, j_k) - \log(P(i_k) - \log P(j_k)$$

We call this algorithm standard path metric-list (SPM-list) decoding.

Though approached in a different way, the above joint decoding method is very similar to the method of LVA based decoding of a concatenated code described in the Section 3.1. The two convolutional coders and the list decoders form the inner coding and decoding stages (we can consider two convolutional coders as a single convolutional coder with the input (i, j)). The path selection unit in our scheme is equivalent to the outer (error detecting) decoder. Here, the equivalent list size is $M \cdot M = M^2$ and any candidate (i_1^L, j_1^L) that does not satisfy the multiple description block code would have $P(i_1^L, j_1^L)/(P(i_1^L)\ P(j_1^L))$ equal to zero and therefore not selected by the path selection unit. Out of the candidates that satisfy the outer error detection multiple description block code, the one with the lowest cost is selected as the output in our scheme. The only exception is that the outer code probabilities are also included in the cost function. However, some modifications will be introduced into the above joint decoding algorithm later to improve the performance and to address some practical issues.

The basic challenge in this joint decoding scheme is selecting a limited number of potentially good description sequence pairs. The main drawback of the above scheme is that it does not consider inter-description correlation in any way in the list decoders and therefore the best sequence pairs may not be included in the final list. One way of improving the "quality" of the list is to use JPM decoding described in Chapter 2 in the list decoders. The JPM-MAP and JPM-ML decoders exploit the correlation presents in the outputs of the convolutional coders. This correlation may be much weaker than the inter-description correlation as suggested by the weaker performance of the JPM algorithms compared to the joint trellis decoding in case of an i.i.d. source (see Figure 2.6). Therefore it can be expected that the integration of the list decoding would improve the performance of JPM decoding. In fact, as shown by the experimental results, this achieves an appreciable improvement. The resulting joint decoding schemes are called JPM-MAP-list algorithm and JPM-ML-list algorithm respectively. The JPM-MAP-list algorithm works as follows:

- List Viterbi decoders 1 and 2 use the path metrics $\log P(i_1^L|r_1^L, s_1^L)$ and $\log P(j_1^L|r_1^L, s_1^L)$ to find M best estimates for i_1^L and j_1^L respectively. From Section 2.22, for an i.id source, the corresponding branch metrics bm_{1k} and bm_{2k} of respective decoders are given by

$$\begin{aligned} bm_{1k} &= \log P(r_k, s_k|v_k) + \log P(i_k) \\ bm_{2k} &= \log P(r_k, s_k|w_k) + \log P(j_k) \end{aligned}$$

  where k denotes the time.

- From the resulting $M \times M$ sequence pairs (i_1^L, j_1^L) the one that minimizes the cost function

$$\begin{aligned} C &= -\{\log P(r_1^L|i_1^L) + \log P(s_1^L|j_1^L) + \log P(i_1^L, j_1^L)\} \\ &= -\sum_{k=1}^{L} \log(P(r_k|i_k) + \log P(s_k|j_k) + \log P(i_k, j_k) \end{aligned}$$

  is selected by the path selection unit as the output. It should be clear that minimizing the above cost function is equivalent to maximizing the right hand side of (3.2)

The JPM-ML-list algorithm is very similar to the JPM-MAP-list algorithm above except that the path metrics used are $\log P(r_1^L, s_1^L|i_1^L)$ and $\log P(r_1^L, s_1^L|j_1^L)$. Accordingly, the priori probability terms $\log P(i_k)$ and $\log P(j_k)$ do not appear in the branch metrics.

The computational complexity of the LVA based decoding increases approximately linearly with the list size. The number of computations in the branch metric calculation will not be increased by the list decoding. However, the list decoding will have an M fold increase in the add, compare and select computations involved in extending and selecting the survivors. At the path selection unit, there are approximately $3M^2L$ additions and M^2 comparisons (assuming that the log probability terms are available in lookup tables). If the number of states in the state space is much larger than the list size the effect of the computations in the path selection unit on the overall computational complexity is negligible. Therefore, the LVA based joint decoding will have approximately M fold increase in the computational complexity for small list sizes.

3.3 Experimental Results

LVA based joint decoding is experimentally evaluated for an i.i.d. Gaussian source of unit variance. The MDSQ is designed as described in the Chapter 1 with $\mu_0 = 1$ and $\mu_1 = \mu_2 = 0.005$. The index assignment used in this experiment is same as that used in the experiment in Chapter 2 (which is given in Figure 2.8). Each description consists of 3 bits and therefore there are 64 elements in the IA matrix out of which 21 are assigned. Descriptions are convolutional coded with the (4,3,2) code (rate 3/4 and constraint length 5) obtained from [21]. The list decoders are designed based on the parallel LVA in [22]. All the priori probabilities are evaluated empirically using a training sequence. We use the same figure of merit, the average signal to distortion ratio defined in Section 2.3. In the experiments, we do not reject invalid sequence pairs because, it is found that at high error probabilities and small list sizes, almost all the candidate pairs in the list have one or more invalid index pairs (an index pair that is not assigned in the IA matrix). Therefore, each such invalid index pair is given a sufficiently small non zero probability value to ensure that the selected sequence pair would have a smaller number of such index pairs.

Figure 3.3 shows the performance variation of LVA based joint decoding schemes with the channel error probability for a list size of 4. The SPM-list decoding achieves a maximum gain of around 2.2 dB relative to separate MAP decoding described in Chapter 2. A higher performance gain is achieved by JPM based list decoding where the maximum gain is around

5.5 dB for the JPM-MAP-list decoding relative to separate MAP decoding. However, we have to understand the fact that a part of this improvement is due to the superiority of the performance of JPM decoding over separate decoding itself. The performance gain achieved explicitly by list decoding is shown in Figures 3.4, 3.5 and 3.6. The JPM-MAP-list decoding has the best performance gain for all the list sizes. It is observed that JPM-ML-list decoding can even be worse than SPM-list decoding at high error probabilities. This is because separate MAP decoding beats JPM-ML decoding at high error probabilities as seen in Figure 2.6. Again, the (explicit) performance gain of the JPM-ML-list decoding is less than that of the SPM-list decoding for the same list size. In all the cases, the performance improves with the list size as expected. The highest performance is bounded above by that of the optimum joint trellis decoding. It is evident that the incremental performance gain decreases rapidly as the list size is increased. But the computational complexity increases at least linearly with the list size and hence a suitable list size has to be decided considering the performance gain and the complexity. We see that both SPM-list and JPM-MAP-list decodings achieve a reasonably good compromise between the performance gain and the computational complexity at list size 4.

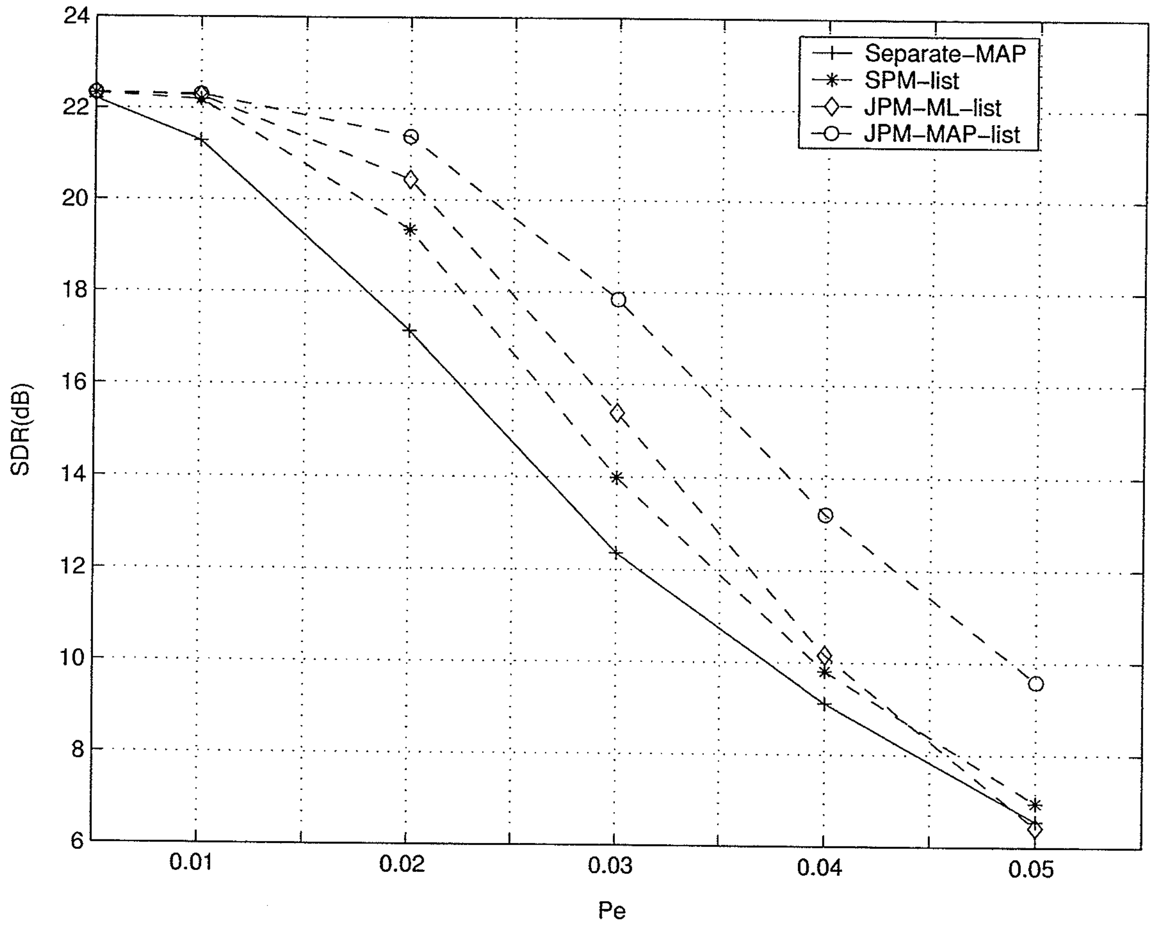


Figure 3.3: Performance of LVA based joint decoding for list size 4.

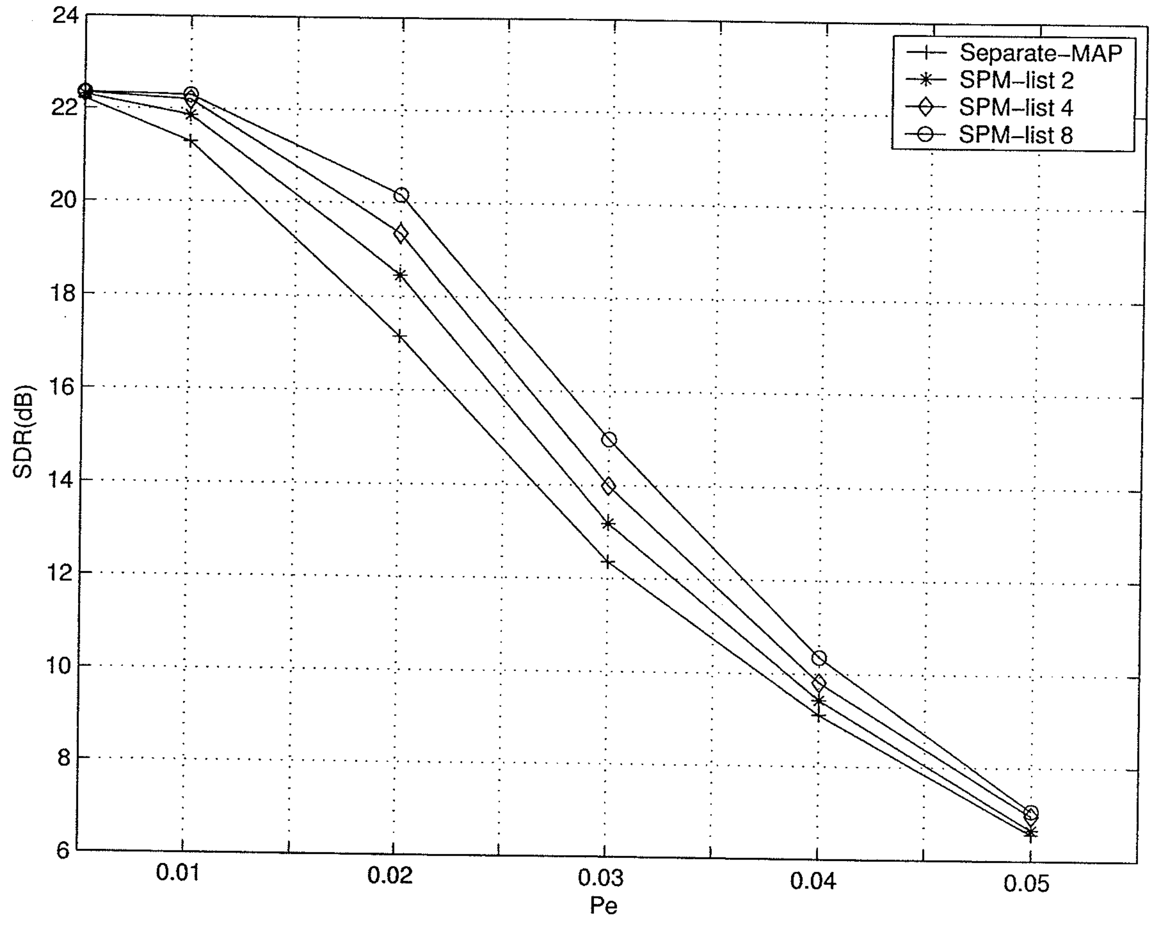


Figure 3.4: Performance variation of SPM-list decoding with the list size.

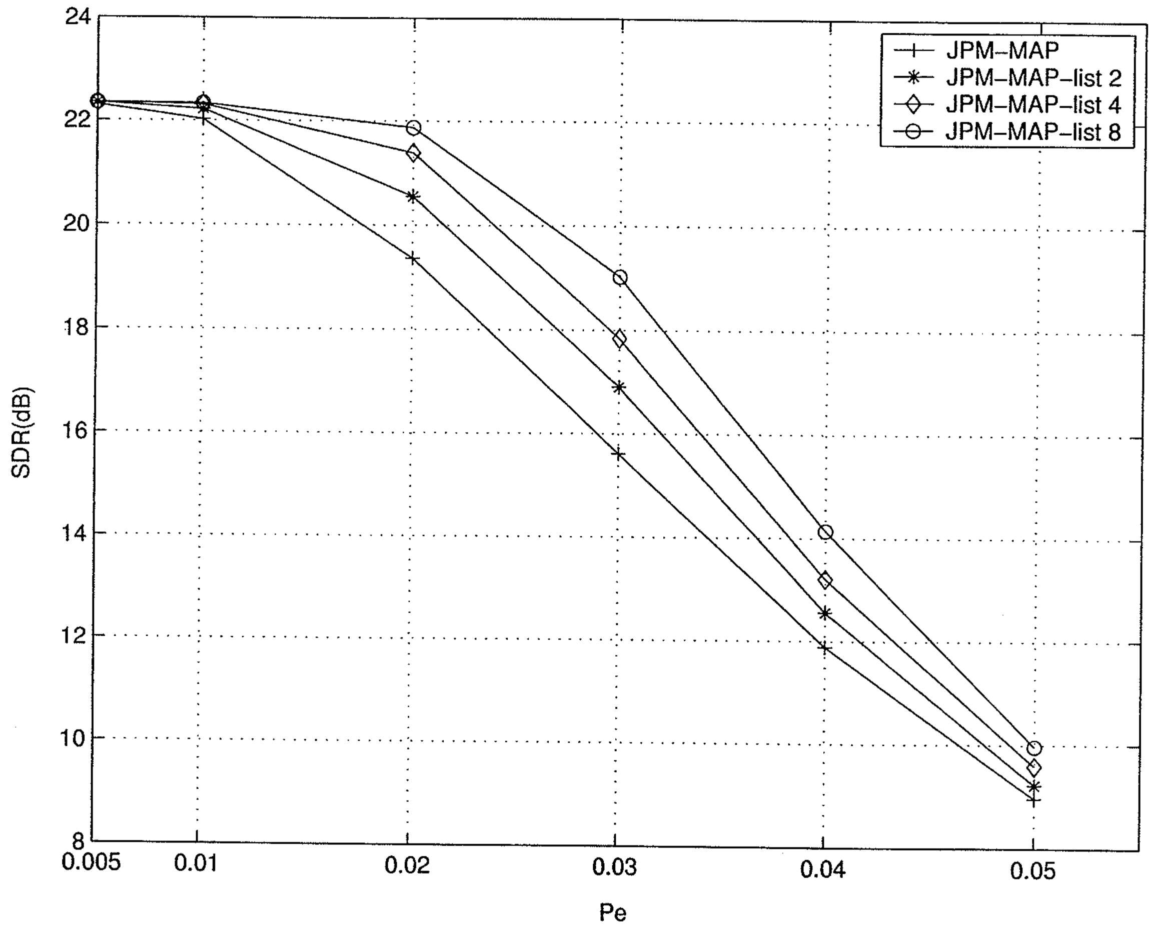


Figure 3.5: Performance variation of JPM-MAP-list decoding with the list size.

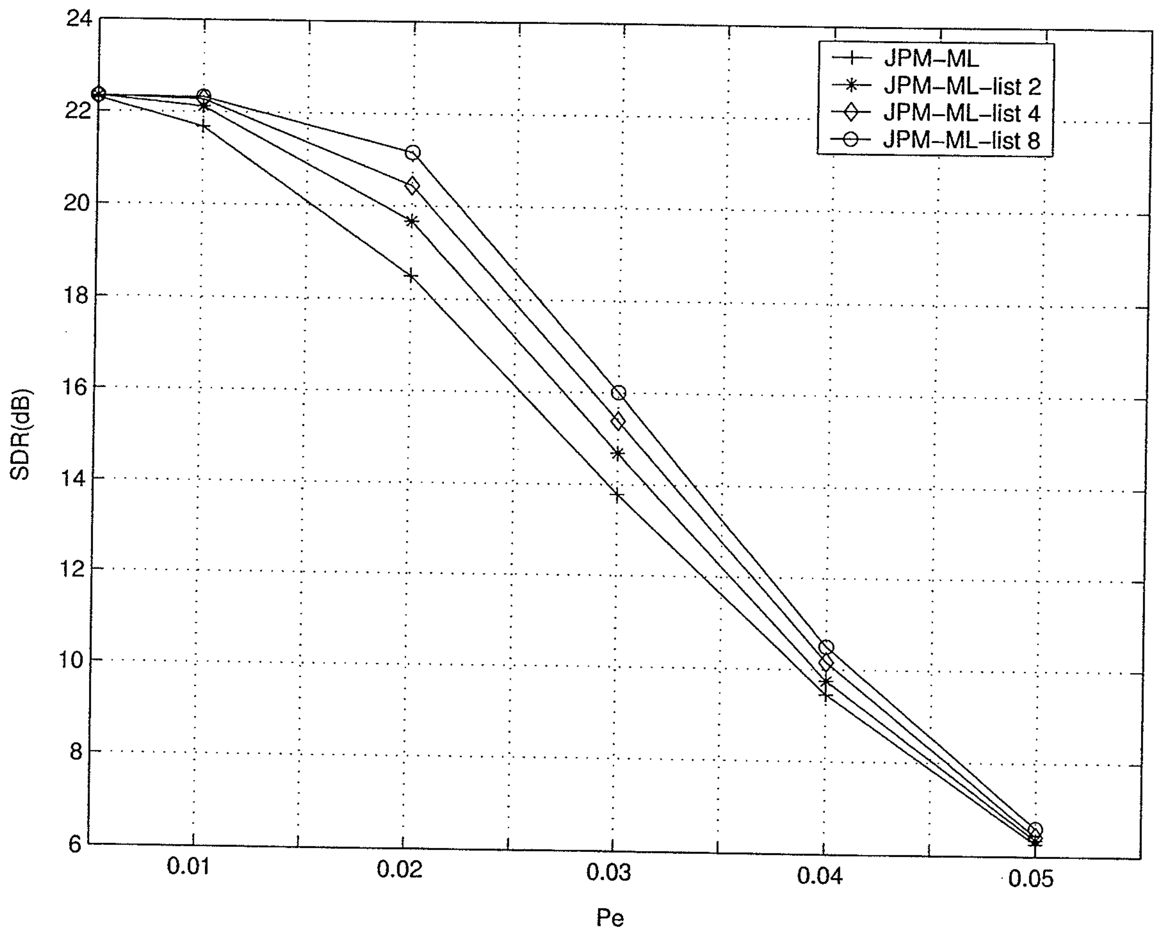


Figure 3.6: Performance variation of JPM-ML-list decoding with the list size.

Chapter 4

Bit Level Joint-Trellis Decoding

As pointed out in Chapter 2, joint trellis decoding is an optimum sequence estimation scheme. It is seen in Chapter 2 (see Figure 2.6) that this scheme even outperforms the iterative decoding scheme presented in [15]. The main drawback of the scheme is its high complexity. As explained in Chapter 2, One cause of the increased complexity is description level encoding and decoding. The joint trellis decoding scheme in [14] requires a one-to-one correspondence between the description alphabet and the convolutional encoder input alphabets and this would result in a large state complexity for large description alphabets. More importantly, given that the fraction of the elements assigned in the IA matrix is constant, the computational complexity of the scheme increases exponentially with the rate (in bits/source sample) of each of the MD encoder output channel. These causes of the complexity can be eliminated by bit level encoding and bit level joint trellis (BLJT) decoding. Bit level encoding here means the channel coding with a rate 1/2 convolutional encoder (possibly with puncturing to achieve the required code rate) and bit level decoding refers to the decoding of the channel code with bit level decisions rather than description level decisions. In other words, the input to the convolutional encoder is considered as a binary source with a particular correlation structure. As explained in Section 4.3, BLJT decoding achieves a significant reduction of complexity over description level joint trellis (DLJT) decoding. Our purpose here is to extend the description level correlation to the

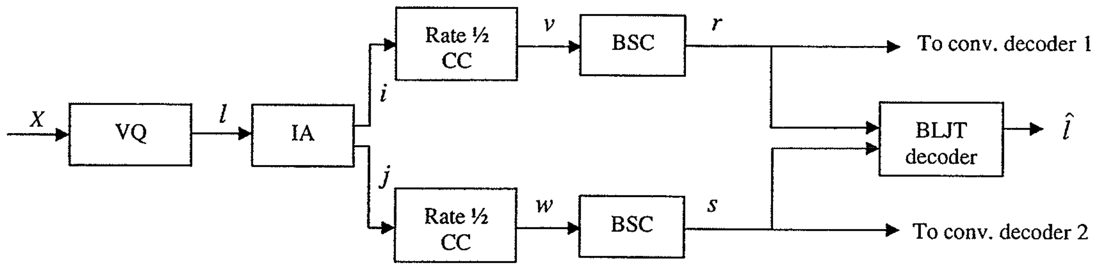


Figure 4.1: Two channel MDQ system with bit level joint trellis decoding.

bit level so that it can be used in a simpler bit level decision decoding scheme without weakening the performance.

Figure 4.1 shows a two channel MDQ system with rate 1/2 convolutional coding and BLJT decoding. The source vector $\mathbf{x}$ is vector quantized and the resulting index l is assigned to b bit indices i=($i(1)$,$i(2)$,...,$i(b)$) and j=($j(1)$,$j(2)$,...,$j(b)$), where $i(k)$ and $j(k), k \leq b$ are the k-th bits of the binary representation of the indices i and j. The binary words are then parallel to serial converted to obtain the input bit stream to the convolutional encoder. Let v=($v(1)$,$v(2)$,...,v(b)) and w=($w(1)$,$w(2)$,...,w(b)) be the output of the convolutional encoders for description i and j where the (two bit) symbols $v(k)$ and $w(k)$ are the outputs for the input bits $i(k)$ and $j(k)$ respectively. Channel coding is done in blocks and when only one codeword is available at the receiver it is decoded by the corresponding convolutional decoder. On the other hand, when both codewords are available they are decoded by the joint decoder.

4.1 Joint Trellis

The joint trellis is obtained by combining the individual trellises of the convolutional coders as described in Chapter 2. Two convolutional coders are considered as a one super convolutional coder with input (i, j) and output (v, w). A state S^J of the joint trellis is given by the pair of states (S_1, S_2) of the individual encoders. Let convolutional coders 1 and 2 have constraint lengths ν_1 and ν_2 respectively. If i and j can take any value then there would be

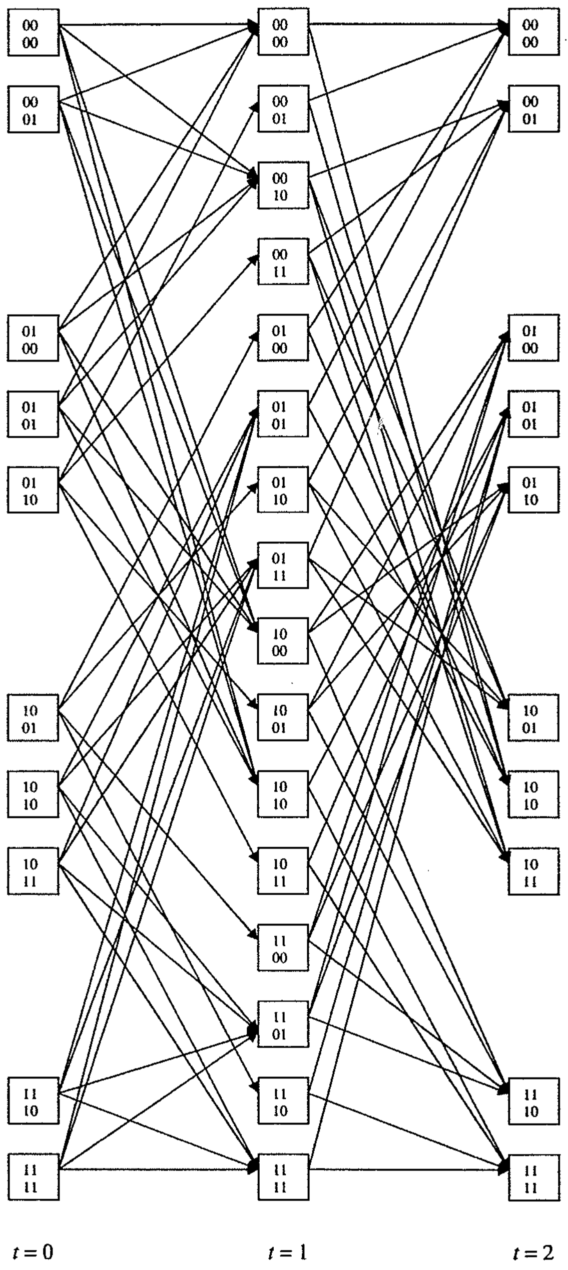


Figure 4.2: A section of the joint trellis for 2 bits/description MDQ with IA given in Figure 4.3 and rate 1/2 convolutional encoders of constraint length 2. Time $t = 0$ corresponds to the beginning of a description period. Transition from $t = 0$ to $t = 1$ is caused by the lower order bits of the two descriptions and the transition from $t = 1$ to $t = 2$ is caused by the higher order bits.

0	1		
2	3	5	
	4	6	7
		8	9

Figure 4.3: An index assignment for two channel MDQ of rates $R_1 = R_2 =$ 2 bits/description.

$2^{\nu_1+\nu_2}$ states in the joint trellis. However, the values that i and j can take are restricted by the index assignment and hence, when not all the elements of the IA matrix are assigned, some of the states will have zero probability. Therefore, the number of states in the joint trellis with non zero probability will be less than $2^{\nu_1+\nu_2}$. Again, due to the same reason, the bit patterns that can be contained in the two shift registers of the convolutional encoders will change with time and hence the trellis will have a periodic time varying structure with the periodicity equal to the number of bits used per description. Figure 4.2 shows a section of the joint trellis for a 2 bits/description MDQ and rate 1/2 convolutional coding with constraint length 2. The corresponding IA matrix is given in Figure 4.3. A state is labelled with the (binary) content of the convolutional coder shift registers. The two bits of the first row in the state label gives the content of the encoder 1 shift register and the second row gives the content of the encoder 2 shift register. We assume natural binary representation of the descriptions.

4.2 MAP and ML Sequence Estimation

Let $l_1^L = (l_1, l_2, \ldots, l_L)$ be the index sequence for the source sequence $\mathbf{x}_1^L = (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_L)$ and let (i_1^L, j_1^L), (v_1^L, w_1^L) and (r_1^L, s_1^L) be corresponding description sequence, channel input sequence and channel output sequence respectively. We are interested in using the Viterbi algorithm to obtain optimum sequence estimate for l_1^L. For example, the MAP estimate of the sequence l_1^L is given by

$$l_1^{*L} = \arg\max_{l_1^L} P(l_1^L | r_1^L, s_1^L).$$

From (2.11), (2.12) and (2.13)

$$\arg\max_{l_1^L} P(l_1^L | r_1^L, s_1^L) = \arg\max_{(i_1^L, j_1^L)} P(r_1^L, s_1^L, i_1^L, j_1^L)$$

As explained in Section 2.2, to apply the Viterbi algorithm in this case, we would have to use a monotonic function of $P(i_1^L, j_1^L, r_1^L, s_1^L)$ as the path metric which can be expressed as a summation of branch metrics. From (2.14), (2.15) and (2.16)

$$P(r_1^L, s_1^L, i_1^L, j_1^L) = \prod_{k=1}^{L} P(r_k | v_k) P(s_k | w_k) P(i_1^L, j_1^L) \tag{4.1}$$

To simplify this further, we have to assume a model for the source. For example, if the process $\{x_k\}$ is i.i.d, then the process $\{(i_k, j_k)\}$ is also i.i.d and the term $P(i_1^L, j_1^L)$ can be expanded as

$$P(i_1^L, j_1^L) = \prod_{k=1}^{L} P(i_k, j_k) \tag{4.2}$$

However, we are working with a bit level trellis and therefore (4.1) and (4.2) has to be expanded into bit level. We know

$$\begin{aligned} P(i_k, j_k) &= P\big(i_k(b), j_k(b), i_k(b-1), j_k(b-1), \ldots, i_k(1), j_k(1)\big) \\ &= \prod_{n=1}^{b} P_{ij}(k, n) \end{aligned} \tag{4.3}$$

where

$$\begin{aligned} P_{ij}(k,n) &= P\big(i_k(n), j_k(n) | i_k(n-1), j_k(n-1), \ldots, i_k(1), j_k(1)\big) \\ &= \frac{P\big(i_k(n), j_k(n), i_k(n-1), j_k(n-1), \ldots, i_k(1), j_k(1)\big)}{P\big(i_k(n-1), j_k(n-1), \ldots, i_k(1), j_k(1)\big)} \\ &= \frac{\sum_{(i_t, j_t) \in \ell_1} P(i_t, j_t)}{\sum_{(i_t, j_t) \in \ell_2} P(i_t, j_t)} \end{aligned} \tag{4.4}$$

Here $\ell_1 = \ell(k,n)$ is the set of all possible index pairs (i_t, j_t) whose first n bits are equal to those of the (i_k, j_k). I.e.

$$i_t(q) = i_k(q) \qquad \forall\, q \leq n$$

and

$$j_t(q) = j_k(q) \qquad \forall\, q \leq n$$

Similarly, $\ell_2 = \ell(k, n-1)$ is the set of all possible index pairs (i_t, j_t) whose first $n-1$ bits are equal to those of the (i_k, j_k). Therefore, for an i.i.d source, (4.1) can be expanded into bit level terms as

$$P(r_1^L, s_1^L, i_1^L, j_1^L) = \prod_{k=1}^{L} \prod_{n=1}^{b} P\big(r_k(n)|v_k(n)\big)\; P\big(s_k(n)|w_k(n)\big)\; P_{ij}(k,n) \tag{4.5}$$

Now, we define our path metric as $\log P(r_1^L, s_1^L, i_1^L, j_1^L)$. Then

$$\begin{aligned}\log P(r_1^L, s_1^L, i_1^L, j_1^L) &= \sum_{k=1}^{L}\sum_{n=1}^{b} \log P\big(r_k(n)|v_k(n)\big) \;+ \log P\big(s_k(n)|w_k(n)\big) \\ &\qquad + \log P_{ij}(k,n) \\ &= \sum_{k=1}^{L}\sum_{n=1}^{b} (bm_{kn})_{iid-MAP} \end{aligned} \tag{4.6}$$

where

$$(bm_{kn})_{iid-MAP} = \log P\big(r_k(n)|v_k(n)\big) + \log P\big(s_k(n)|w_k(n)\big) + \log P_{ij}(k,n) \tag{4.7}$$

is the branch metric that we are going to use in the Viterbi algorithm.

As mentioned in the discussion in Section 2.2, the first-order Markov process is a widely used model for some practical sources. Therefore, let's consider the case where $\{\mathbf{x}_k\}$ is first-order Markov. In this case, making the same assumption as in Section 2.2 that the process $\{l_k\}$ also is first-order Markov, we have

$$P(i_1^L, j_1^L) = P(i_1, j_1) \prod_{k=2}^{L} P(i_k, j_k | i_{k-1}, j_{k-1})$$

Here, the term $P(i_1, j_1)$ can be expanded into bit level terms by using (4.3). The term $P(i_k, j_k | i_{k-1}, j_{k-1})$ can be expanded as

$$\begin{aligned} P(i_k, j_k | i_{k-1}, j_{k-1}) &= P\big(i_k(b), j_k(b), i_k(b-1), j_k(b-1), \ldots, i_k(1), j_k(1) | i_{k-1}, j_{k-1}\big) \\ &= \prod_{n=1}^{b} P'_{ij}(k,n) \qquad \text{for } 2 \le k \le L \end{aligned} \tag{4.8}$$

where

$$\begin{aligned} P'_{ij}(k,n) &= P\big(i_k(n), j_k(n) | i_k(n-1), j_k(n-1), \ldots, i_k(1), j_k(1), i_{k-1}, j_{k-1}\big) \\ &= \frac{P\big(i_k(n), j_k(n), i_k(n-1), j_k(n-1), \ldots, i_k(1), j_k(1), i_{k-1}, j_{k-1}\big)}{P\big(i_k(n-1), j_k(n-1), \ldots, i_k(1), j_k(1), i_{k-1}, j_{k-1}\big)} \\ &= \frac{\sum_{(i_t,j_t),(i_{t-1},j_{t-1}) \in \ell'_1} P\big((i_t, j_t), (i_{t-1}, j_{t-1})\big)}{\sum_{(i_t,j_t),(i_{t-1},j_{t-1}) \in \ell'_2} P\big((i_t, j_t), (i_{t-1}, j_{t-1})\big)} \qquad \text{for } 2 \le k \le L \end{aligned} \tag{4.9}$$

Here

$$\ell'_1 = \ell'(k,n)$$

where $l'(k,n)$ is the set of all possible two consecutive index pairs $\{(i_{t-1}, j_{t-1}), (i_t, j_t)\}$ such that

$$(i_{t-1}, j_{t-1}) = (i_{k-1}, j_{k-1})$$

and

$$(i_t(q), j_t(q)) = (i_k(q), j_k(q)) \qquad \forall\, q \le n$$

Similarly

$$\ell'_2 = \ell'(k, n-1)$$

Let

$$P'_{ij}(1,n) = P_{ij}(1,n)$$

Following the same steps as (4.5)–(4.7) we obtain

$$(bm_{kn})_{mkv-MAP} = \log P\big(r_k(n)|v_k(n)\big) + \log P\big(s_k(n)|w_k(n)\big) + \log P'_{ij}(k,n) \tag{4.10}$$

the branch metric for the MAP sequence estimation in case of a first-order Markov source.

Before going into details of the application of the Viterbi algorithm in MAP sequence estimations described above, it is required to analyze the properties of the process $\{(i_k(n), j_k(n)\}$ which is the input to the super convolutional encoder. In the case of an i.i.d source, the index pair (i_k, j_k) is independent of previous index pairs. Therefore,

$$P(i_k(n), j_k(n)|i_k(n-1), j_k(n-1), \ldots, i_k(1), j_k(1), i_{k-1}(b), j_{k-1}(b), \ldots) = \\ P(i_k(n), j_k(n)|i_k(n-1), j_k(n-1), \ldots, i_k(1), j_k(1)) \tag{4.11}$$

Hence, the super convolutional coder input process $\{(i_t(n), j_t(n))\}$ is a non stationary Markov process whose order p changes periodically. Here, the largest value of p is $b-1$. If we further assume that $\{(i_k, j_k)\}$ is first-order Markov, then

$$P\big((i_k, j_k)|(i_{k-1}, j_{k-1}), (i_{k-2}, j_{k-2}), \ldots\big) = P\big((i_k, j_k)|(i_{k-1}, j_{k-1})\big)$$

Therefore

$$P(i_k(n), j_k(n)|i_k(n-1), j_k(n-1), \ldots, i_k(1), j_k(1), i_{k-1}(b), j_{k-1}(b), \ldots) = \\ P(i_k(n), j_k(n)|i_k(n-1), j_k(n-1), \ldots, i_{k-1}(1), j_{k-1}(1)) \tag{4.12}$$

In this case too, $(i_t(n), j_t(n))$ is a non-stationary Markov process whose order p changes periodically. The largest value of p is $2b-1$.

To use the Viterbi algorithm in the MAP sequence estimations described above, it is required that the (super) state process $\{S_t^J\}$ be first-order Markov [18]. The state S_t^J represents the bits that are stored in the super convolutional coder shift registers at time t. To understand the requirements that has to be satisfied for $\{S_t^J\}$ to be first-order Markov, let's consider a generic state process $\{s_t\}$ where

$$s_t = (a_{t-1}, a_{t-2}, \ldots, a_{t-m_1})$$

Assume that $\{a_t\}$ is a Markov process of order m_2. Then

$$P(a_t|a_{t-1}, a_{t-2}, \ldots) = P(a_t|a_{t-1}, a_{t-2}, \ldots, a_{t-m_2}) \tag{4.13}$$

Now, we want to find the relationship between m_1 and m_2 when $\{s_t\}$ is first-order Markov. If $\{s_t\}$ is first-order Markov, then

$$P(s_t|s_{t-1}, s_{t-2}) = P(s_t|s_{t-1})$$

Therefore

$$P(a_{t-1}, \ldots, a_{t-m_1}|a_{t-2}, \ldots, a_{t-m_1}, a_{t-m_1-1}, \ldots) = P(a_{t-1}, \ldots, a_{t-m_1}|a_{t-2}, \ldots, a_{t-m_1-1})$$

Hence

$$P(a_{t-1}|a_{t-2}, \ldots, a_{t-m_1}, a_{t-m_1-1}, \ldots) = P(a_{t-1}|a_{t-2}, \ldots, a_{t-m_1-1}) \tag{4.14}$$

But, $\{a_t\}$ is m_2-th order Markov. Therefore

$$m_1 \geq m_2 \tag{4.15}$$

Now, we see that $\{S_t^J\}$ is analogous to $\{s_t\}$ and $\{a_t\}$ is analogous to the super convolutional coder input process $\{(i_k(n), j_k(n))\}$. Let m be the memory of the super convolutional coder which is analogous to m_1 above. Therefore, applying the result in (4.15), the requirement that has to be satisfied for $\{S_t^J\}$ to be first-order Markov is

$$m \geq p_{max} \tag{4.16}$$

where p_{max} is the largest value of the order p of the non stationary Markov process $\{(i_k(n), j_k(n))\}$. We saw that, when the MDQ output is i.i.d., $p_{max} = b - 1$ and therefore, the requirement that has to be satisfied to apply the Viterbi algorithm in MAP sequence estimation which we described earlier in this section is

$$m \geq b - 1. \tag{4.17}$$

When the MDQ output is first-order Markov, the requirement is

$$m \geq 2b - 1. \tag{4.18}$$

In joint trellis maximum likelihood (JT-ML) decoding, the sequence (l_1^L) that maximizes $P(r_1^L, s_1^L | l_1^L)$ is found by searching through the joint trellis that is described in section 4.1. We see that

$$\begin{aligned} P(r_1^L, s_1^L | l_1^L) &= P(r_1^L, s_1^L | i_1^L, j_1^L) \\ &= P(r_1^L, s_1^L | v_1^L, w_1^L) \\ &= \prod_{k=1}^{L} P(r_k, s_k | v_k, w_k) \\ &= \prod_{k=1}^{L} \prod_{n=1}^{b} P\big(r_k(n), s_k(n) | v_k(n), w_k(n)\big) \end{aligned} \tag{4.19}$$

let's take $\log P(r_1^L, s_1^L | l_1^L)$ as the path metric for this ML sequence estimation. The corre-

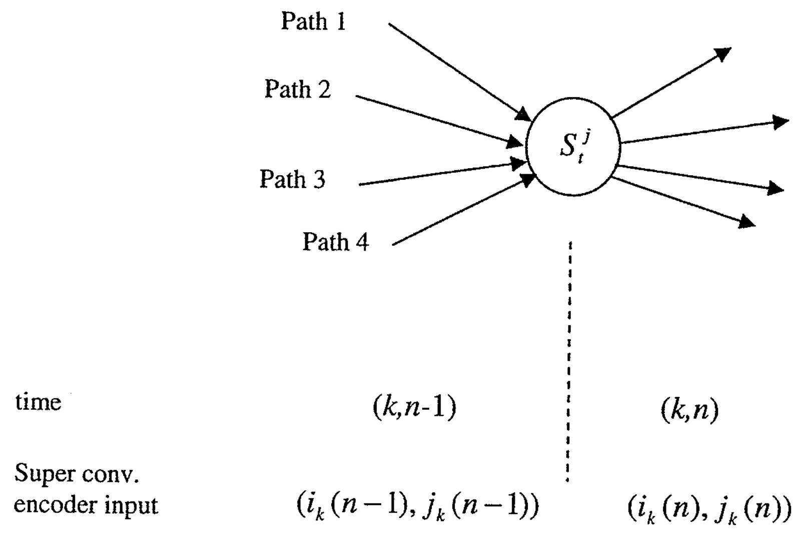


Figure 4.4: A state in the joint trellis at the beginning of the n-th bit period of the k-th index pair (i_k, j_k) .

sponding branch metric is

$$(bm_{kn})_{ML} = \log P\big(r_k(n), s_k(n)|v_k(n), w_k(n)\big)$$

Now we want to find whether there is any requirement that has to be satisfied to apply the Viterbi algorithm in ML sequence estimation as described above. Figure 4.4 shows a state S_t^J in the joint trellis at the beginning of the n-th bit period of the k-th index pair (i_k, j_k) (we denote this time position by $t = (k, n)$). The Viterbi algorithm involves finding the path that has the highest partial path metric (sum of the branch metric along the path) for each state. Without loss of generality, assume that the path 1 has the largest partial path metric (PPM) out of all the paths arriving at state S_t^J. In the Viterbi algorithm, this path is selected as the survivor path for the state S_t^J and the other paths are eliminated. The argument here is that the globally best path (path with the largest path metric) cannot begin with any of the eliminated paths. Otherwise, we can replace its initial segment (path segment up to state S_t^J) with path 1 to get another path that has a higher path metric (this is a contradiction). One of the underlying assumptions in the above argument is that the suggested replacement procedure results in a valid path. However, in the case of a joint trellis, this may not always be true since some paths are invalid depending on the IA. The requirement for above argument to hold in case of a joint trellis is that the previous bits $(i_k(1), j_k(1), i_k(2), j_k(2), \ldots, i_k(n-1), j_k(n-1))$ of the current description be same for all the paths arriving at the same state. Otherwise, the paths arriving at the same state may have different sets $\{(i_k(n), j_k(n)\}$ of possible (next) input bit pairs (depending on the IA) and therefore they may extend along different branches. If this happens, it is clear that, the above path segment replacement process may result in an invalid path. To guarantee that the previous bits of the current description be always the same for all the paths arriving at the same state, it is required that

$$m \geq b - 1 \tag{4.20}$$

Therefore, this is the requirement that has to be satisfied in applying the Viterbi algorithm for JT-ML sequence estimation.

4.3 Computational Complexity Comparison Between Description Level Decoding and Bit Level Decoding

BLJT decoding has a reduced computational complexity compared to the DLJT decoding for same convolutional coder constraint lengths. This reduction is very similar to that achieved by a rate k/n convolutional code obtained by puncturing a rate $1/2$ mother code over a rate k/n k-ary input convolutional code [27]. In a rate k/n k-ary convolutional code, there will be 2^k branches arriving at each state in the trellis and therefore it would require $2^k - 1$ comparisons per state per k input bits in finding the survivor and hence the computational complexity is exponential in k. On the other hand, a rate k/n punctured binary input convolutional code would require only k comparisons per state per k input bits. It is clear that this achieves a significant reduction in computational complexity for large k. The punctured convolutional code would have the same computational complexity if the survivors are found after each k input bits. The complexity reduction is due to the elimination of a path after each bit period for each state, which would otherwise grow exponentially as in a binary tree. The same argument applies to the complexity comparison between BLJT decoding and DLJT decoding. In case of DLJT decoding, number of comparisons needed at a state is $2^{2b}q$ where q is the fraction of elements used in the IA matrix and therefore, if we assume q to be constant, the computational complexity of the algorithm increases exponentially with the description size. In case of BLJT decoding, there is no easy way to find the exact number of comparisons required since the trellis has a time varying structure which depends on the particular index assignment. In this case, there are at most 3 comparisons required at a state in the joint trellis per input symbol $(i_t(m), j_t(m))$. For the same constraint length, if survivors are found after each description period, decoding with the bit level joint trellis would in general have the same order of computational complexity as the DLJT decoding. Similar to the punctured convolutional code, the BLJT decoding achieves a significantly reduced computational complexity by eliminating paths after each symbol period, which would otherwise multiply to produced more paths. However, this multiplication can be quaternary, ternary, binary or even unary depending on the particular path, time position and the index assignment.

It is shown in Section 4.2 that optimum sequence estimation with the Viterbi algorithm requires the convolutional coder memory to be greater than a parameter related to b. However, the experimental results show that the BLJT decoding achieves a significant performance improvement over the separate decoding even when the encoder does not satisfy this memory requirement.

4.4 Experimental Results

BLJT decoding algorithm presented in this chapter is experimentally evaluated for different settings to illustrate the performance improvement achieved and the effect of various parameters on the performance. Simulations are done for an i.i.d Gaussian source with unit variance and a Gauss Markov source given by

$$x_t = 0.9x_{t-1} + w_t$$

where $\{w_t\}$ is a zero mean unit variance i.i.d Gaussian process. This process is a simple model for practical sources such as speech and images. The source is MD coded by a two channel MDSQ designed as described in Chapter 1. The parameters used in the design are $\mu_0 = 1$ and $\mu_1 = \mu_2 = 0.05$. The simulations are done for the binary symmetric channel model. In case of BLJT MAP decoding, the priori probabilities are calculated empirically using a training sequence. In the experiments, we consider only the case where both the codewords are received by the decoder since, joint decoding affects only the performance of the central decoder. We use as the figure of merit, the average signal to distortion ratio

$$SDR = EX^2/E(X - \hat{X})^2$$

where $\hat{X}$ is the reconstruction of X at the decoder.

Figures 4.5 and 4.6 show the variation of the SDR with channel error probability for an i.id. source and GM source respectively. JT-ML and JT-MAP denote BLJT ML and BLJT MAP decoding respectively. Separate-MAP and Separate-ML denote the MAP and ML sequence estimation with separate decoding (as explained in Section 2.3) respectively. The index assignment in Figure 2.8 is used in the MDSQ. Each description is of size 3 bits and 3 diagonals are used in the IA matrix. Descriptions are encoded with a rate 3/4 convolutional code obtained by puncturing the (2,1,6) (i.e. 2 outputs, 1 input and memory 6) convolutional code in Table 12.4 (b) in [19]. Note in this case that the memory requirements (4.17) and (4.20) are satisfied for the i.i.d. source. The memory requirements are satisfied in case of the GM source too, If we approximate the MDSQ output as a first-order Markov process. At low bit error probabilities, the channel code corrects almost all the errors and therefore no noticeable difference exists in the performance between separate decoding and joint decoding. However, as the bit error probability is increased, BLJT decoding achieves a significant performance improvement over separate decoding. Particularly, for both the sources, BLJT decoding schemes correct almost all the errors in the received codewords up to the channel bit error probability of 0.02 whereas the separate decoding loses about 8 dB in performance. The MAP decoding performs better than the ML decoding as expected. In case of the i.i.d source this is due to the redundancy present in the form of non uniformity in the probabilities of the descriptions. MAP decoding achieves a considerably higher performance than ML decoding for the GM source since it uses the redundancy present in the form of strong temporal correlation in addition to the non uniformity of the description probabilities. The strength of the BLJT ML decoding scheme is that it does not require any knowledge of the source and in general would perform well irrespective of the source that is being transmitted.

Figures 4.8 and 4.9 show the performance of the BLJT decoding and separate decoding for a system with rate 4 bits/description MDSQ. The MDSQ outputs are channel coded using rate 3/4 convolutional coders obtained by puncturing the (2,1,2) code obtained from Table 12.4 (b) in [19]. The IA matrix used in the MDSQ is given in Figure 4.7. In the IA matrix 5 diagonals are assigned and hence 74 out of 256 elements are used. Therefore, the fraction of elements used in the IA matrix in this case is close to that of the systems in Figures 4.5 and 4.6. Note in this case that (4.17) and (4.20) requires the convolutional encoder memory of at least 3 bits for the iid source. For GM source, under the assumption that MDSQ output is first-order Markov, the convolutional encoder memory requirement is at least 7 bits. Therefore, the BLJT decoding may not result in optimum sequence estimates in this case. Nonetheless, experimental results show that the scheme achieves a significant performance improvement.

Finally, Figures 4.11, 4.12 and 4.13 show the variation of the performance of BLJT decoding with the number of diagonals used in the IA matrix. In all three cases, rate 3 bits/description MDSQ is used with rate 3/4 convolutional code obtained by puncturing the (2,1,3) convolutional code obtained from Table 12.4 (b) in [19]. Out of the 64 elements in the IA matrix, 15, 21 and 34 elements corresponding to 2, 3 and 5 diagonals are used in the

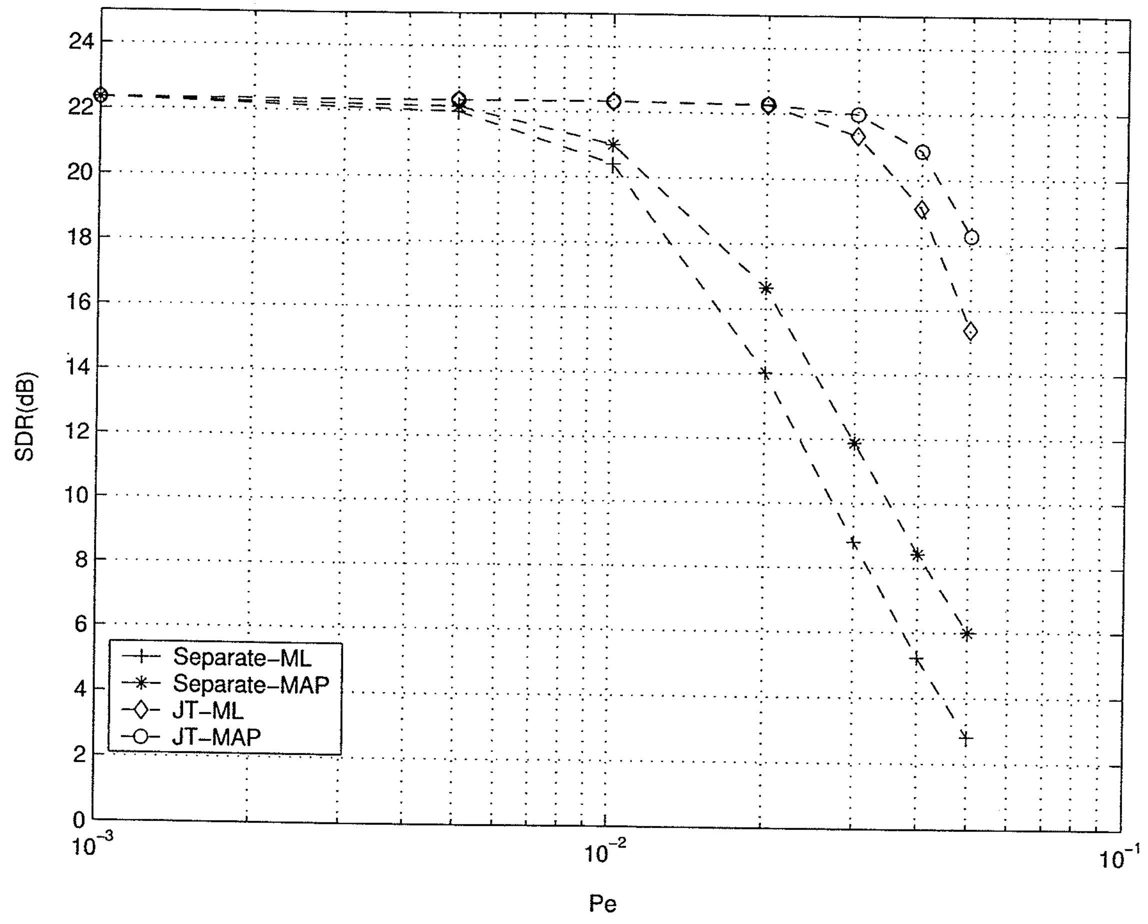


Figure 4.5: Performance of BLJT decoding and separate decoding for i.i.d Gaussian source, $R_1 = R_2 = 3$ bits/description, number of diagonals used = 3, convolutional coder memory= 6.

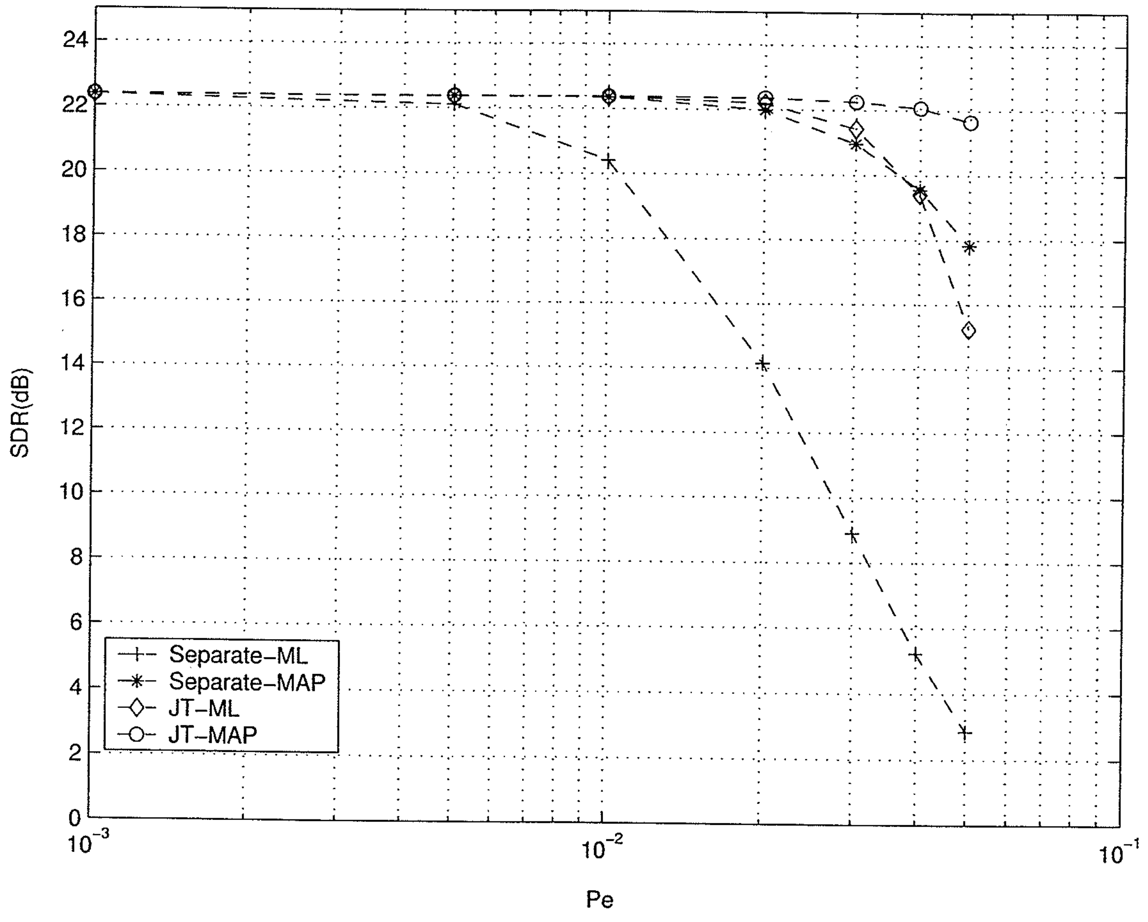


Figure 4.6: Performance of BLJT decoding and separate decoding for GM source, $R_1 = R_2 = 3$ bits/description, number of diagonals used = 3, convolutional coder memory= 6.

0	1	3													
2	5	6	8												
4	7	10	12	14											
	9	11	15	17	19										
		13	16	20	21	23									
			18	22	25	26	28								
				24	27	30	32	34							
					29	31	35	37	39						
						33	36	40	41	43					
							38	42	45	46	48				
								44	47	50	52	54			
									49	51	55	57	59		
										53	56	60	61	63	
											58	62	65	66	68
												64	67	70	72
													69	71	73

Figure 4.7: Index assignment for the simulation in Figure 4.8.

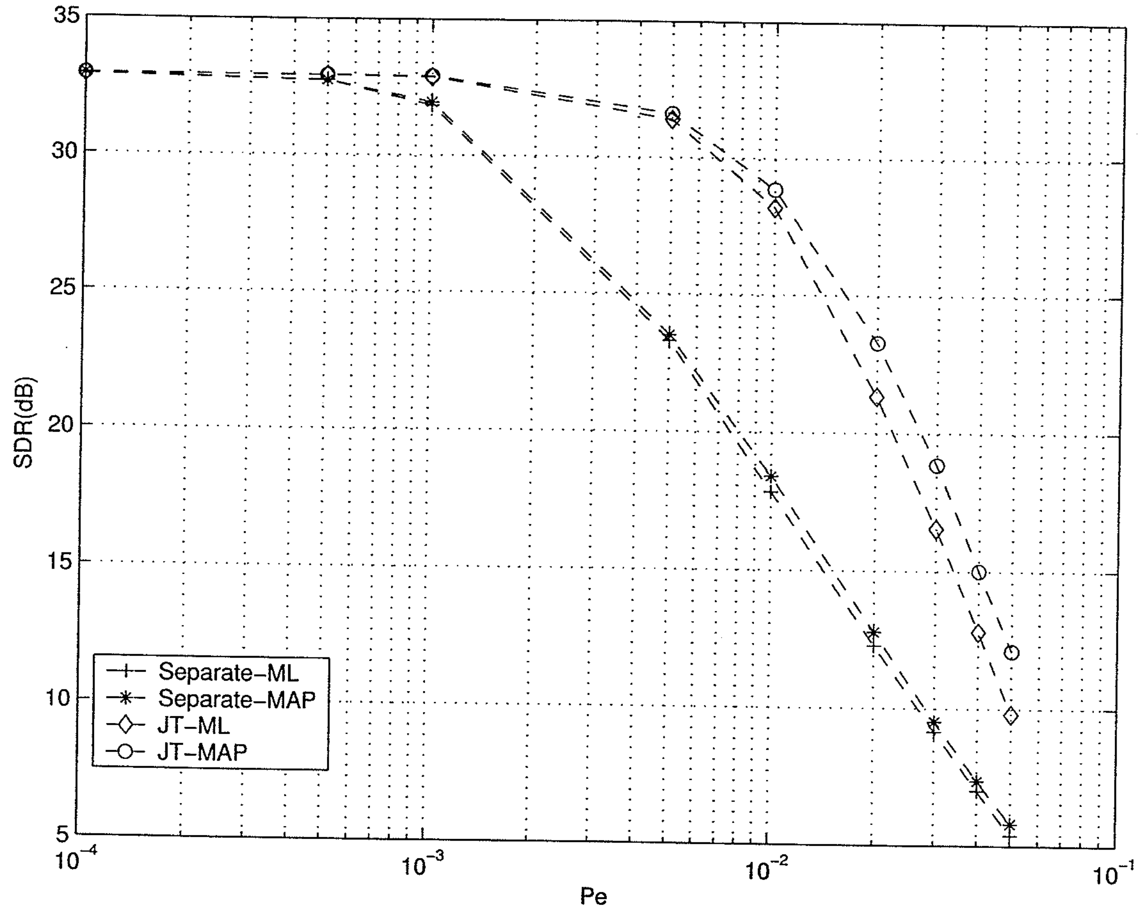


Figure 4.8: Performance of BLJT decoding and separate decoding for iid Gaussian source, $R_1 = R_2 = 4$ bits/description, number of diagonals used = 5, convolutional coder memory= 2.

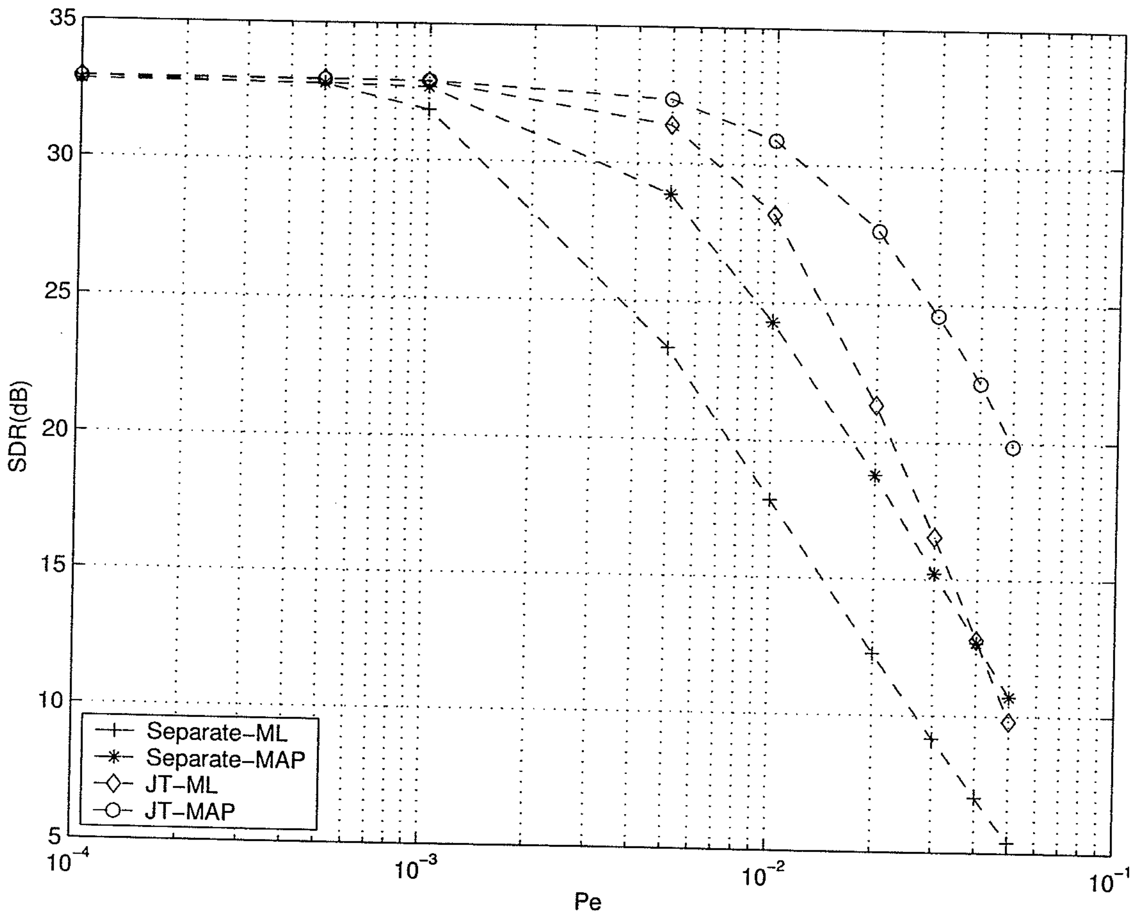


Figure 4.9: Performance of BLJT decoding and separate decoding for GM source, $R_1 = R_2 = 4$ bits/description, number of diagonals used = 5, convolutional coder memory= 2.

0	1						
	2	3					
		4	5				
			6	7			
				8	9		
					10	11	
						12	13
							14

Figure 4.10: Index assignment for the simulation in Figure 4.11.

three systems respectively. The IA matrix in Figures 2.8 and 1.6 are used for the simulation in Figures 4.12 and 4.13 respectively. The IA matrices for the simulations in Figure 4.11 is given in the Figure 4.10. As one would expect, the performance gain of the BLJT decoding algorithm decreases with the decrease of the redundancy in the index assignment.

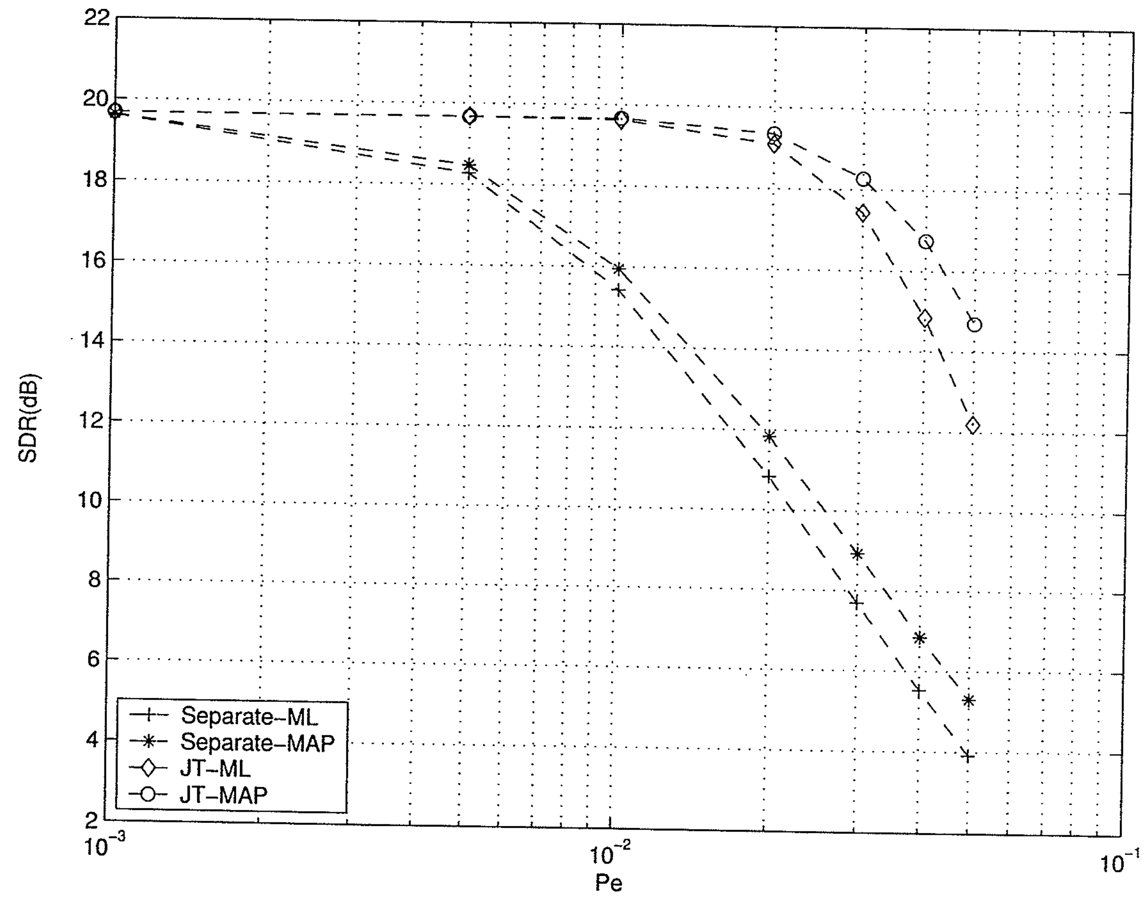


Figure 4.11: Performance of BLJT decoding and separate decoding for iid Gaussian source, $R_1 = R_2 = 3$ bits/description, number of diagonals used = 2, convolutional coder memory= 3.

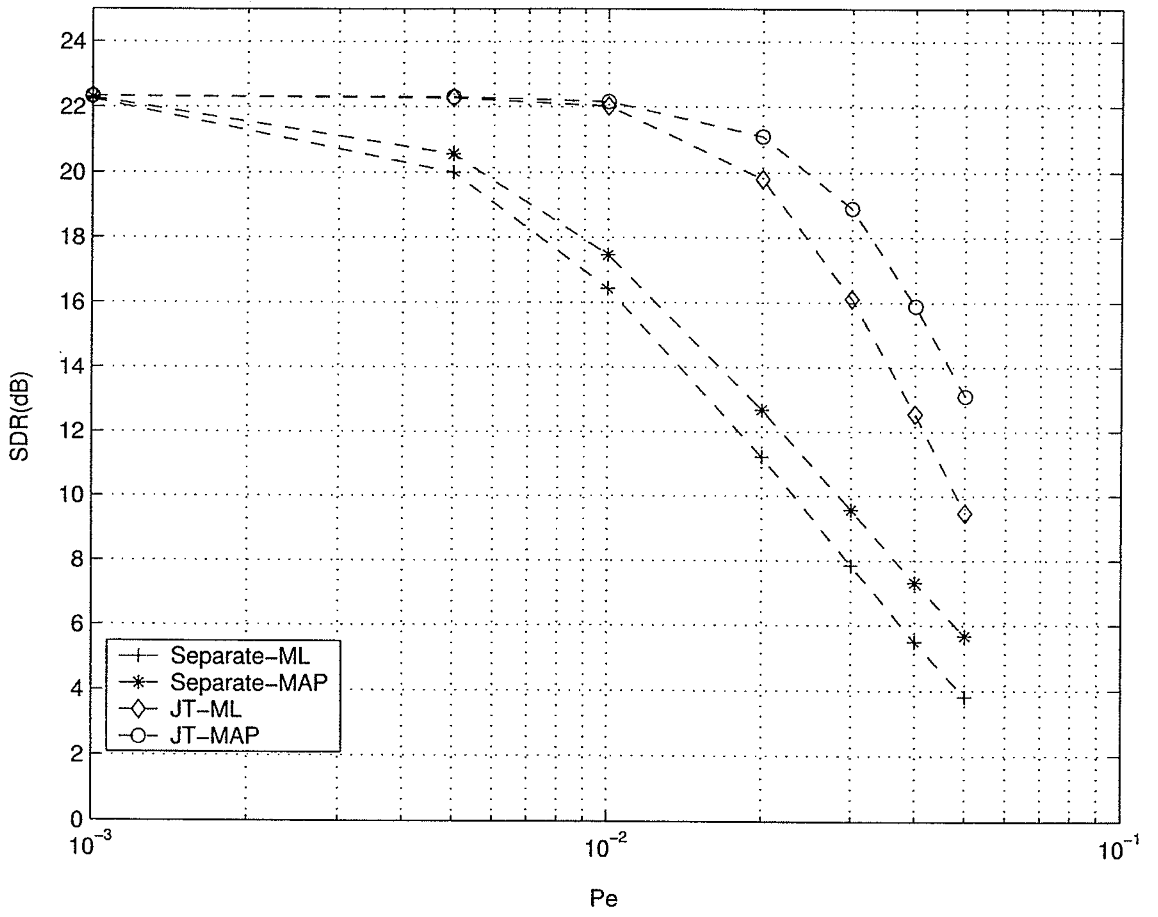


Figure 4.12: Performance of BLJT decoding and separate decoding for iid Gaussian source, $R_1 = R_2 = 3$ bits/description, number of diagonals used = 3, convolutional coder memory= 3.

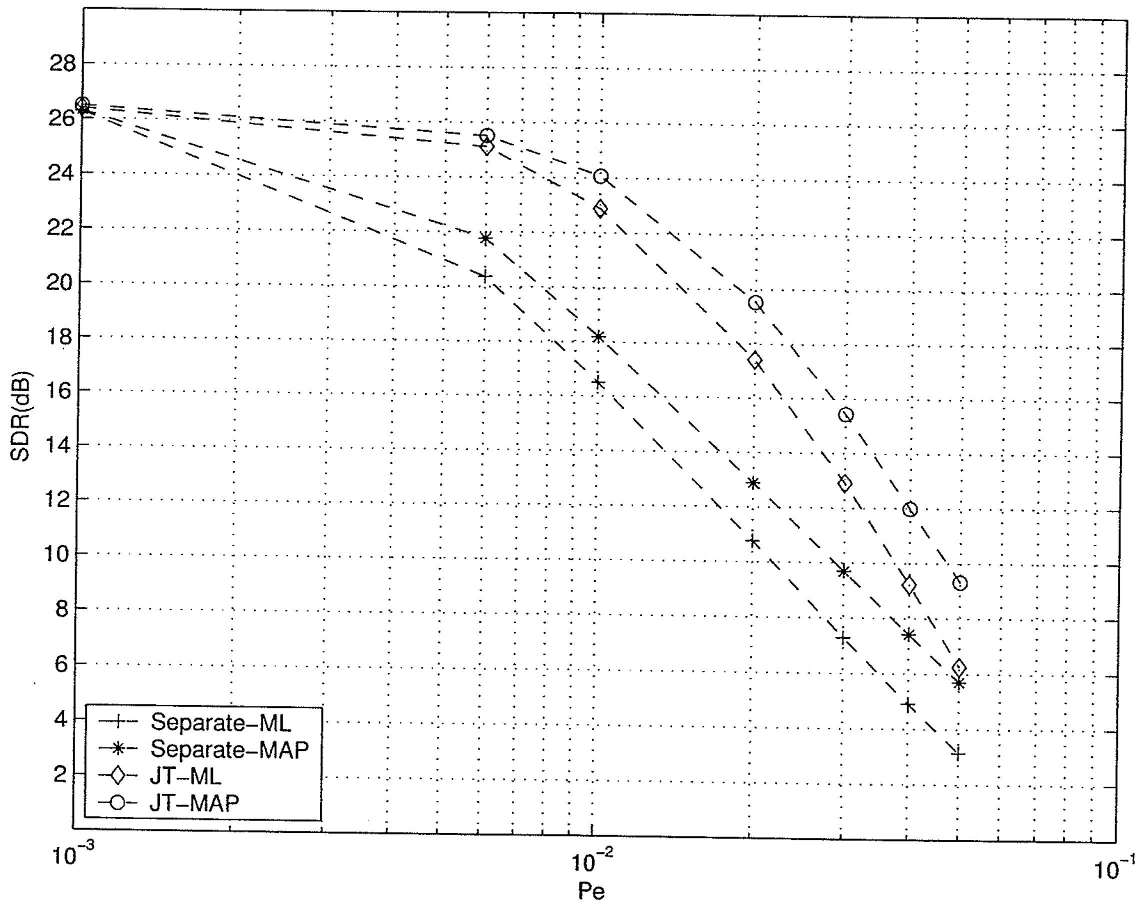


Figure 4.13: Performance of BLJT decoding and separate decoding for iid Gaussian source, $R_1 = R_2 = 3$ bits/description, number of diagonals used = 5, convolutional coder memory= 3.

Chapter 5

Index Re-Mapping

MDC can be viewed as a systematic addition of redundancy to guard against transmission losses such as packet losses. On the other hand, in error control coding, redundancy is added to guard against random bit errors. Broadly viewed, both error control coding and MDC come under the common problem of systematic addition of redundancy to achieve robustness against unreliable transmission. Therefore it can be expected that there may be joint coding schemes to achieve the robustness over both random bit errors and transmission losses which perform better than separate coding. So far, we considered only joint decoding of separately encoded signals. In this chapter, we discuss a simple form of joint encoding, which we call *index re-mapping.*

In Chapters 2 and 4, we saw that joint trellis decoding achieves a significant improvement over separate decoding. It is seen in Figures 4.11, 4.12 and 4.13 that this improvement greatly depends on the number of elements assigned in the IA matrix. To gain more insight into this phenomenon, let us consider an example system consisting of a two channel MDQ of rate $R_1 = R_2 = b$ bits/description followed by a convolutional encoder. Assume that N out of M^2 elements are assigned in the IA matrix, where $M = 2^b$. Let the source sequence $\mathbf{x}_1^L = (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_L)$ be MD coded into index sequence (i_1^L, j_1^L) and let (v_1^L, w_1^L) and (r_1^L, s_1^L) be the channel input sequence and the channel output sequence respectively. Consider the two convolutional encoders as a single super convolutional encoder with input (i_t, j_t). If the symbol (i_t, j_t) has no constraint, there would be $(M \cdot M)^L = M^{2L}$ number of possible length L input sequences and therefore, we would have to consider a similar number of paths in the joint trellis However, the symbol (i_t, j_t) is constrained by the IA and therefore, we would have to consider only N^L paths in the joint trellis. In other words, out of M^{2L} codewords in the super convolutional code, only N^L will be valid in this case. For a given channel output sequence (r_1^L, s_1^L), we select in joint trellis decoding the best estimate $(\hat{v}_1^L, \hat{w}_1^L)$ of the channel input sequence from these N^L sequences. In case of separate decoding, an individual decoder is used on each channel to obtain the best estimate of the channel input sequence from M^L possible sequences (let $\bar{v}_1^L$ and $\bar{w}_1^L$ be the corresponding estimates). Taken together, this is equivalent to selecting a codeword from M^{2L} codewords in the super convolutional code. To make this clearer, an example is considered after introducing some basic concepts in channel coding such as *Hamming distance* and *minimum distance.*

Hamming distance between two symbol sequences is the number of symbol positions in which they differ. Performance of a channel code is determined by its Hamming distance properties. When codewords are farther apart, the channel code has a better error correcting capability. The number of errors q that a channel code can correct is determined by a parameter called minimum distance, denoted by d_{min} which is the minimum distance between any two codewords of the code. Codewords that are most likely to be confused at the channel output (due to channel errors) with a given transmitted codeword are its neighbors at distance d_{min}. q is related to d_{min} by [19]

$$d_{min} \geq 2q + 1.$$

Now, consider the BSC channel model and ML decoding. In this case, the joint trellis decoding is equivalent to finding the sequence $(\hat{v}_1^L, \hat{w}_1^L)$ from the valid codeword set such that $d(r_1^L, \hat{v}_1^L) + d(s_1^L, \hat{w}_1^L)$ is minimum where $d(\mathbf{a}, \mathbf{b})$ represents the Hamming distance between the binary representation of the sequences **a** and **b**. Separate decoding, in this case, involves finding the sequences $\bar{v}_1^L$ and $\bar{w}_1^L$, each from the corresponding M^L possible sequences, such that $d(r_1^L, \bar{v}_1^L)$ and $d(s_1^L, \bar{w}_1^L)$ are minimum. It should be clear that this is equivalent to finding the sequence $(\bar{v}_1^L, \bar{w}_1^L)$ from the full codeword set of the super convolutional coder such that $d(r_1^L, \bar{v}_1^L) + d(s_1^L, \bar{w}_1^L)$ is minimum. Therefore, joint trellis decoding attains an improvement in error correcting capability over separate decoding due to two reasons:

1. Separate Decoding may result in an invalid sequence pair whereas joint trellis decoding always gives a valid sequence pair.

2. In joint trellis decoding, we consider a reduced set of codewords. Consider a transmitted (hence, valid) super codeword $\mathbf{c} = (v_1^L, w_1^L)$. It can be expected that the number of neighbors for $\mathbf{c}$ at distance d_{min} , on the average, is less in the reduced codeword set than that in the full codeword set. Hence, in case of joint trellis decoding, there would be fewer codewords to be confused with $\mathbf{c}$ at the channel output.

The distance properties of the subset of the super convolutional code, which we consider in joint trellis decoding, can be further improved if we have the freedom to to select the N^L codewords from M^{2L} codewords. However, this cannot be done since, the possible codewords are determined by the IA. But, still, we can change the valid codeword set without changing the IA by re-mapping the indices. As an example, we can relabel the indices in the IA in Figure 5.1 to get an equivalent IA shown in Figure 5.2. It is clear that this does not change the basic structure of the MDQ and its properties (such as encoder partition, decoder codebook, central and side distortions etc.).

To demonstrate how the index re-mapping can be used to improve the distance properties of a channel code, let's consider a simple example, where the outputs of a MDQ of rate $R_1 = R_2 = 4$ bits/description are channel coded with a (7,4) Hamming code given in Table 5.1. Assume that the IA given in Figure 5.3 is used. Let the indices i_t and j_t be encoded into codewords $\mathbf{v}_t$ and $\mathbf{w}_t$ respectively. Since we are interested in joint decoding, let's consider the two codewords as a single super codeword $\mathbf{c}_t = (\mathbf{v}_t, \mathbf{w}_t)$. In the given Hamming code, $d_{min} = 3$ and therefore, it is capable of correcting a single bit error in a channel output codeword. Our objective in this case is to see how the index re-mapping can be used to

	0	1	2	3	4	5	6	7
0	0	1	3					
1	2	5	6	8				
2	4	7	10	12	14			
3		9	11	15	17	19		
4			13	16	20	21	23	
5				18	22	25	26	28
6					24	27	30	32
7						29	31	33

Figure 5.1: An index assignment with natural numbering of the indices.

	3	6	2	5	1	4	0	7
1	0	1	3					
7	2	5	6	8				
2	4	7	10	12	14			
6		9	11	15	17	19		
4			13	16	20	21	23	
0				18	22	25	26	28
3					24	27	30	32
5						29	31	33

Figure 5.2: An index assignment obtained by Re-mapping the indices in IA matrix in Figure 5.1.

0	1														
2	3	5													
	4	6	7												
		8	9	11											
			10	12	13										
				14	15	17									
					16	18	19								
						20	21	23							
							22	24	25						
								26	27	29					
									28	30	31				
										32	33	35			
											34	36	37		
												38	39	41	
													40	42	43
														44	45

Figure 5.3: An index assignment for MDQ of rate 4 bits/description.

Table 5.1: The Hamming (7,4) code

Message	Decimal representation of the Message	Codeword
0000	0	0000000
0001	1	0001111
0010	2	0010110
0011	3	0011001
0100	4	0100101
0101	5	0101010
0110	6	0110011
0111	7	0111100
1000	8	1000011
1001	9	1001100
1010	10	1010101
1011	11	1011010
1100	12	1100110
1101	13	1101001
1110	14	1110000
1111	15	1111111

improve the distance properties of the valid codeword set of the super code. There are 256 codewords in the super code out of which only 46 ($N = 46$) are valid. Let $\mathbf{c}_t^{(1)} = (\mathbf{v}_t^{(1)}, \mathbf{w}_t^{(1)})$ and $\mathbf{c}_t^{(2)} = (\mathbf{v}_t^{(2)}, \mathbf{w}_t^{(2)})$ be the two (super) codewords assigned to the distinct index pairs $(i_t^{(1)}, j_t^{(1)})$ and $(i_t^{(2)}, j_t^{(2)})$ respectively. We see that

$$d(\mathbf{c}_t^{(1)}, \mathbf{c}_t^{(2)}) = d(\mathbf{v}_t^{(1)}, \mathbf{v}_t^{(2)}) + d(\mathbf{w}_t^{(1)}, \mathbf{w}_t^{(2)}) \tag{5.1}$$

Therefore, when $i_t^{(1)} \neq i_t^{(2)}$ and $j_t^{(1)} \neq j_t^{(2)}$ (i.e. when the index pairs are not in the same row or column of the IA matrix)

$$\mathbf{v}_t^{(1)} \neq \mathbf{v}_t^{(2)}$$

and

$$\mathbf{w}_t^{(1)} \neq \mathbf{w}_t^{(2)}$$

Therefore

$$d(\mathbf{c}_t^{(1)}, \mathbf{c}_t^{(2)}) \geq 6$$

On the other hand, when the index pairs are in the same row or same column, one of the component codeword of $\mathbf{c}_t^{(1)}$ and $\mathbf{c}_t^{(2)}$ are the same and therefore, $d(\mathbf{c}_t^{(1)}, \mathbf{c}_t^{(2)})$ can be as small as 3. As an example if $i_t^{(1)} = i_t^{(2)}$ then

$$\mathbf{v}_t^{(1)} = \mathbf{v}_t^{(2)}$$

and

$$d(\mathbf{c}_t^{(1)}, \mathbf{c}_t^{(2)}) = d(\mathbf{w}_t^{(1)}, \mathbf{w}_t^{(2)})$$

Therefore, it is clear that the codewords should be assigned to indices in such a manner that $d(\mathbf{w}_t^{(1)}, \mathbf{w}_t^{(2)})$ is as large as possible when $\mathbf{v}_t^{(1)} = \mathbf{v}_t^{(2)}$ and vice versa. In the given Hamming code, for each codeword, there are 7 codewords at distance 3, 7 codewords at distance 4 and one codeword at distance 7. Figure 5.4 shows the codewords that are assigned by the Hamming code in Table 5.1 for natural numbering of the indices. In this case, there are 40 valid (super) codeword pairs with Hamming distance 3. A few examples for such codeword pairs are given in Table 5.2. Figure 5.5 shows codeword assignment after re-mapping the indices in the IA in Figure 5.4 for improving the minimum distance between valid codewords. In this case, there are only 4 pairs of valid codewords with Hamming distance 3. They are given in the Table 5.3. Hamming distance of 36 codeword pairs in Figure 5.4 is increased from 3 to 4 by the index re-mapping. Although this increase in the distance does not improve the error correcting capability of the codewords (Hamming distance of 4 can guarantee only the correction of single bit errors, similar to Hamming distance of 3), the example demonstrates that the distance properties of the valid codeword set can be improved by a proper index re-mapping.

In this example, the index re-mapping was obtained by searching through the (7,4) Hamming code based on the argument that when one of the component codewords is same in two super (valid) codewords, the other component codewords should be as far apart as possible. However, this kind of search is impractical when the description alphabets are large or the channel coding is complex. Therefore, some structured method has to be found for obtaining a good index re-mapping. The main challenge that has to be faced in solving this problem is the non linearity of the valid codeword set. Therefore, some of the well known techniques for designing and analyzing channel codes that are present in the literature cannot be easily applied in this case since, those techniques are valid only for linear codes.

5.1 Simulation Results

To find the performance improvement attainable with index re-mapping in joint trellis decoding, a random search was carried out over 1000 possible index re-mappings. The basic experimental setting is same as that in the simulation for Figure 4.12. From 1000 random index re-mappings, the one giving the best performance improvement was found for each of the decoding schemes separate-ML, JT-ML and JT-MAP and the results are shown in the Figure 5.6. The curves marked as separate-ML, JT-ML and JT-MAP are obtained by using the IA with natural numbering of the indices (same as in the simulation for Figure 4.12).

The MDSQ used in the experiment is of rate $R_1 = R_2 = 3$ bits/description and hence, each description alphabet size is 8. Therefore, there are $8! \times 8! - 1 \simeq 1.6 \times 10^9$ possible index re-mappings. Although, the search is carried out over a small fraction (around $1/1.6 \times 10^6$) of all possible re-mappings, the performance improvement is significant. As explained at the beginning of the chapter, the JT-ML decoding achieves an improvement over separate decoding by only exploiting the improved distance structure in the valid codeword set. On the other hand, the JT-MAP scheme gains an improvement by exploiting both the improved distance structure in the valid codeword set and the redundancy present in the index sequence in the form of non uniform probabilities. Therefore, the improvement caused by the index re-mapping is higher in JT-ML decoding compared to JT-MAP decoding. It can be seen that the performance of JT-ML decoding with index re-mapping even beats that of the JT-MAP decoding with the original IA. The performance of separate decoding also is improved by index re-mapping, since the end to end distortion in a vector quantizer

Table 5.2: Some of the valid (super) codeword pairs in the Figure 5.4 with Hamming distance 3

$(\mathbf{v}_t^{(1)}, \mathbf{w}_t^{(1)})$	$(\mathbf{v}_t^{(2)}, \mathbf{w}_t^{(2)})$
(00000000001111)	(00101100001111)
(00011110000000)	(00011110010110)
(00011110001111)	(00011110010110)
(00011110001111)	(00101100001111)
(00011110010110)	(00101100010110)
(00101100001111)	(00101100010110)
(00101100001111)	(00101100011001)
(00011110010110)	(00110010010110)

Table 5.3: The four valid (super) codeword pairs in the Figure 5.5 with Hamming distance 3

$(\mathbf{v}_t^{(1)}, \mathbf{w}_t^{(1)})$	$(\mathbf{v}_t^{(2)}, \mathbf{w}_t^{(2)})$
(11010011101001)	(00110011101001)
(11010010011001)	(00110010011001)
(11010011101001)	(11010010011001)
(00110011101001)	(00110010011001)

system depends also on the assignment of channel codewords to quantizer output indices as explained in [28]. Finally, the Figure 5.7 shows the performances obtained for the best IA and the worst IA found in a 1000 random index re-mappings. It also shows the performance obtained with the original IA. It is interesting to notice that the performance obtained with the worst IA in this case is considerably better than the original IA. Therefore, in this case, it can be concluded that an index re-mapping chosen at random would considerably improve the performance with a high probability.

		0000000	0001111	0010110	0011001	0100101	0101010	0110011	0111100	1000011	1001100	1010101	1011010	1100110	1101001	1110000	1111111
		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0000000	0	0	1														
0001111	1	2	3	5													
0010110	2		4	6	7												
0011001	3			8	9	11											
0100101	4				10	12	13										
0101010	5					14	15	17									
0110011	6						16	18	19								
0111100	7							20	21	23							
1000011	8								22	24	25						
1001100	9									26	27	29					
1010101	10										28	30	31				
1011010	11											32	33	35			
1100110	12												34	36	37		
1101001	13													38	39	41	
1110000	14														40	42	43
1111111	15															44	45

Figure 5.4: Codewords assigned by the Hamming code in Table 5.1 for natural numbering of the indices.

		0000000	0001111	0110011	0111100	1010101	1011010	1100110	1101001	0011001	0010110	0100101	0101010	1000011	1001100	1110000	1111111
		0	1	6	7	10	11	12	13	3	2	4	5	8	9	14	15
0000000	0	0	1														
0001111	1	2	3	5													
0110011	6		4	6	7												
0111100	7			8	9	11											
1010101	10				10	12	13										
1011010	11					14	15	17									
1100110	12						16	18	19								
1101001	13							20	21	23							
0011001	3								22	24	25						
0010110	2									26	27	29					
0100101	4										28	30	31				
0101010	5											32	33	35			
1000011	8												34	36	37		
1001100	9													38	39	41	
1110000	14														40	42	43
1111111	15															44	45

Figure 5.5: Codewords assigned by the Hamming code in Table 5.1 for an IA obtained by re-mapping the indices in Figure 5.4.

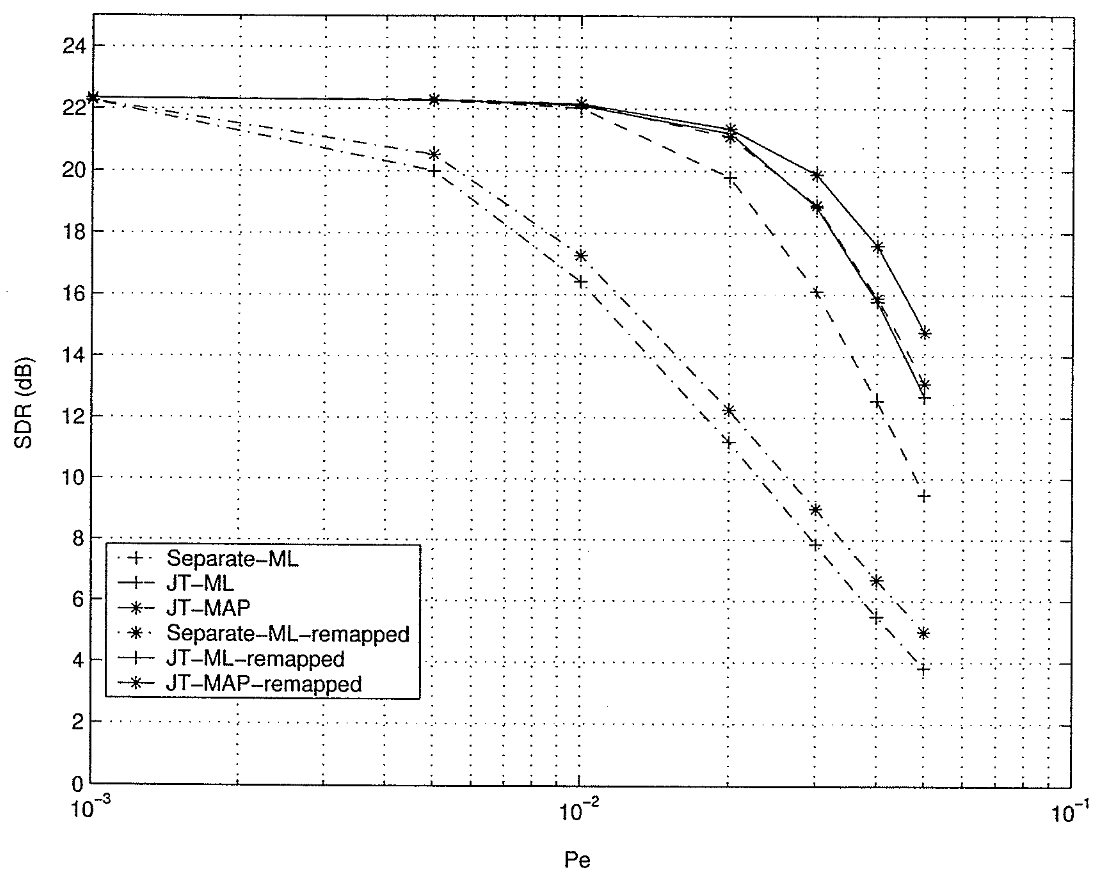


Figure 5.6: Performance improvement obtained by index re-mapping. The basic experimental setting is same as that in Figure 4.12.

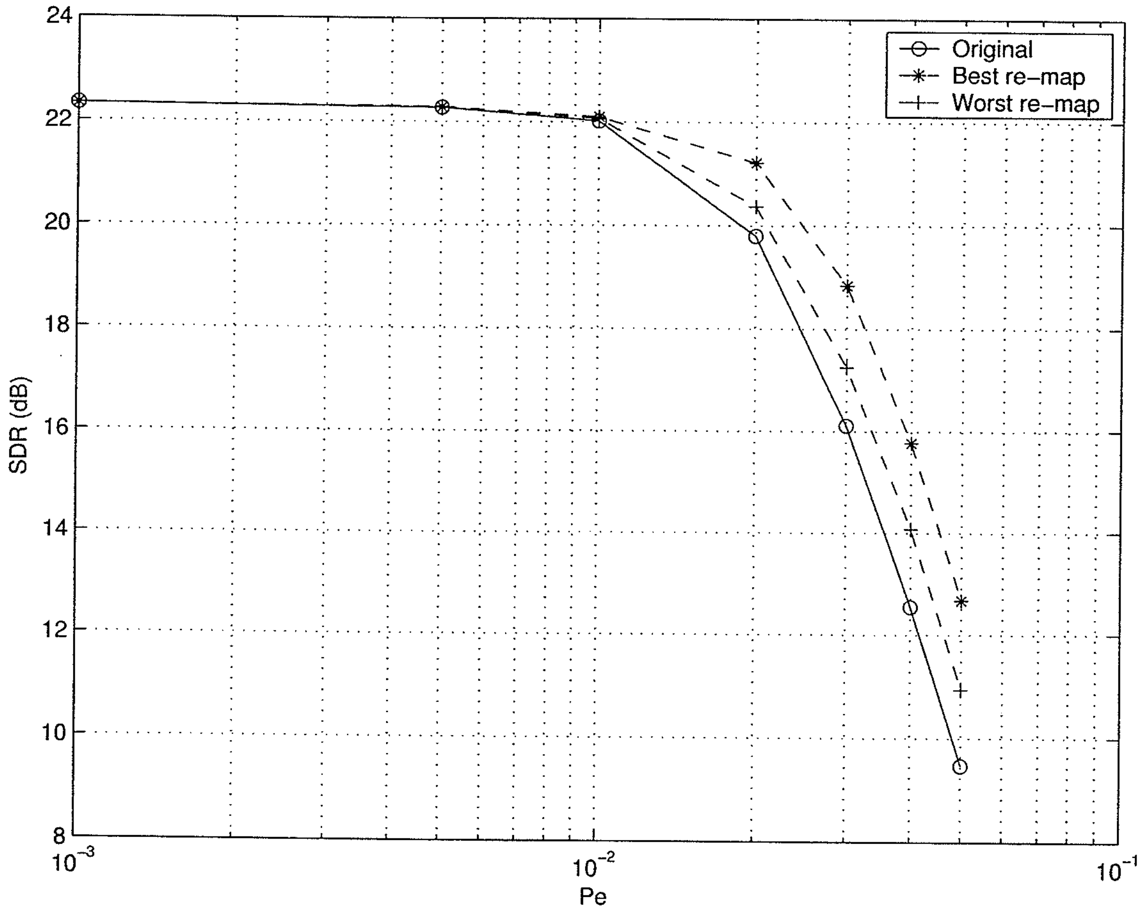


Figure 5.7: Performance obtained for the best and the worst in 1000 random index re-mappings. The figure also shows the performance for the original IA. The basic experimental setting is same as that in Figure 4.12.

Chapter 6

Conclusions and Suggestions for Further Study

In this thesis, we focussed on the problem of joint source channel coding of multiple description. We presented a list Viterbi algorithm based joint decoder and a bit level extension to the joint trellis decoder in [14]. We also introduced the problem of joint source channel encoding with a simple, yet efficient joint encoding procedure known as index re-mapping.

The LVA based joint decoding scheme presented in Chapter 3 is capable of achieving a significant improvement over separate decoding and JPM decoding at a moderate complexity increase. This scheme is based on the idea that an MD code can be viewed as a sort of error detection block code and hence the channel coding of an MD code gives rise to a concatenated coding. We considered the scenario where MD coder outputs are convolutional encoded before the transmission. At the receiver, we used list Viterbi decoding of the inner (convolutional) code. As mentioned in Chapter 3, it would be possible to use other soft decoding algorithms such as BCJR algorithm and SOVA in decoding the inner code. Therefore in a further study, it would be interesting to investigate how the other soft decoding algorithms can be used in a joint decoder and what the achievable improvement are.

Bit level joint trellis decoding scheme presented in Chapter 4 eliminates the requirement of one to one correspondence in the description alphabet and the convolutional coder input alphabet in the description level joint trellis decoding algorithm in [14]. It also achieves a considerable reduction in the computational complexity compared to description level joint trellis decoding, which is very similar to the reduction in computational complexity achieved by a punctured binary input convolutional code over a same rate non binary input convolutional code of the same constraint length. It was shown that bit level joint trellis decoding algorithm is optimal if the convolutional coder memory is greater than a particular value determined by the number of bits per description and the source temporal correlation. Experimental results showed that the bit level joint trellis decoding achieves a significant performance gain even when the optimality requirement is not satisfied. However, similar to description level joint trellis decoding, bit level joint trellis decoding suffers from the problem of high complexity caused by the trellis combining.

In Chapter 6 we presented the idea of joint source channel encoding of multiple descriptions. With index re-mapping we showed that a joint encoding procedure would considerably improve the performance compared to separate encoding. In the experiment in Chapter 6, we obtained an improved index re-mapping by a random search. The problem of finding a structured procedure for obtaining a good index re-mapping has to be addressed in a future work. There would be many other ways to achieve joint encoding and this would be an interesting topic for a future research work.

Appendix A

The Viterbi Algorithm

The Viterbi algorithm (VA) provides an efficient way of finding optimal state sequence estimate of a finite-state discrete time Markov process. This appendix discusses the application of VA in decoding a convolutional code. Before presenting the VA, some concepts in convolutional coding will be briefly introduced.

A convolutional encoder can be represented by a shift register circuit. For example, Figure A.1 shows a (3,1,2) (i.e. 3 bit output, 1 bit input and memory 2) convolutional encoder. The output $\mathbf{v}_t = (v_t^{(0)}, v_t^{(1)}, v_t^{(2)})$ of this encoder is related to its input u_t by

$$v_t^{(0)} = u_t \oplus u_{t-1}$$

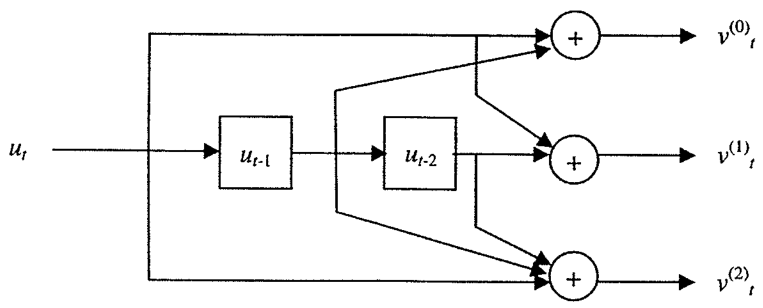


Figure A.1: A (3,1,2) convolutional encoder.

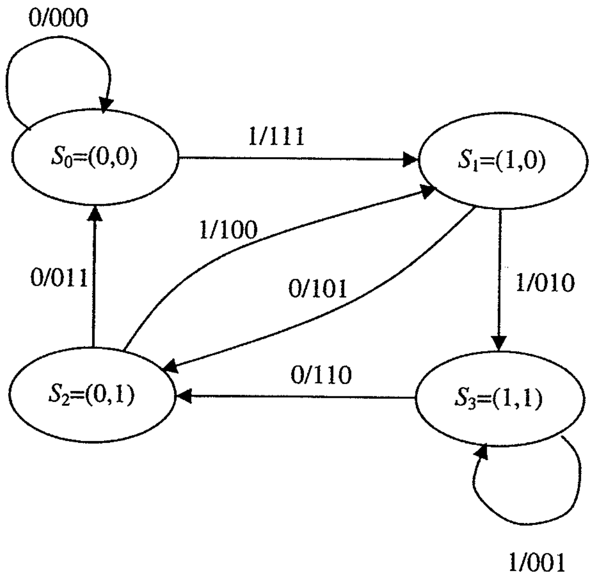


Figure A.2: State diagram for the convolutional encoder in Figure A.1.

$$v_t^{(1)} = u_t \oplus u_{t-2}$$

and

$$v_t^{(2)} = u_t \oplus u_{t-1} \oplus u_{t-2}$$

where, $u_t, u_{t-1}, u_{t-2}, v_t^{(0)}, v_t^{(1)}$ and $v_t^{(2)}$ are binary digits and $\oplus$ represents the modulo-2 addition. This convolutional encoder can be represented by the state diagram in Figure A.2. The state of the convolutional encoder at time t is the content (u_{t-1}, u_{t-2}) of its shift register. Assume that the process starts and terminates at the *all zero* state S_0. Each branch of the state diagram is labelled with the input bit causing the transition and the corresponding three output bits. Figure A.3 shows a time expansion of the state diagram in Figure A.2, which is called as trellis. In the trellis, each node corresponds to a distinct state at a given time and each branch represents a transition to some new state at the next instant of time.

Now, assume that an information sequence $\mathbf{u}_1^L = (\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_L)$, is encoded by a (n, k, m) convolutional encoder. Here, $\mathbf{u}_t$ is the k bit information symbol at time t. To

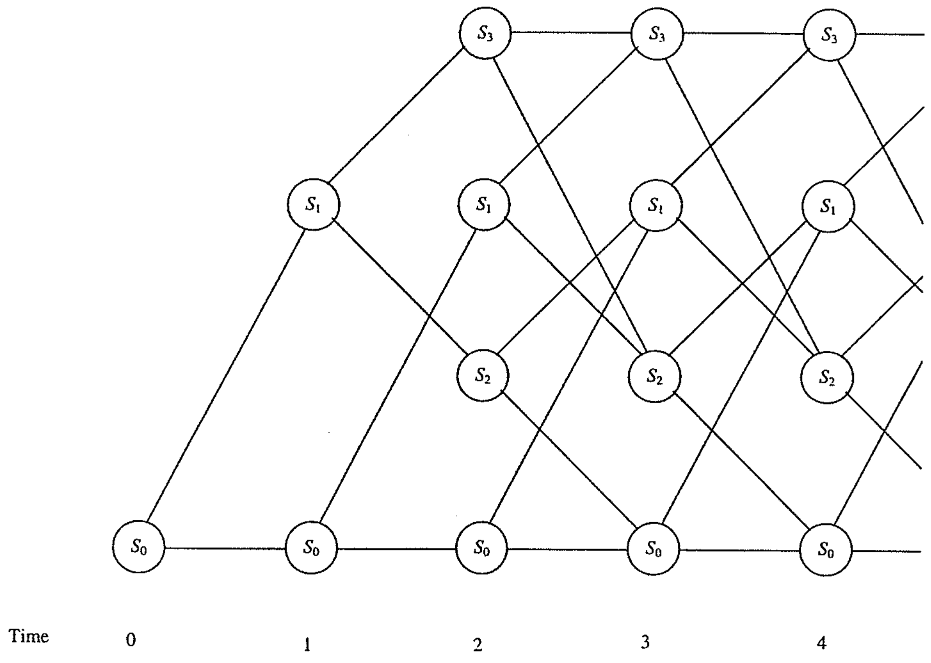


Figure A.3: Trellis diagram for the convolutional encoder in Figure A.1.

drive the encoder into the all zero state at the termination, it is required to append another m symbols to the information sequence such that each such symbol consist of all zero bits. Therefore, the convolutional encoder input sequence is given by $\mathbf{u}_1^{L+m} = (\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_{L+m})$, where $\mathbf{u}_t = (0, 0, \ldots, 0)$ when $t > L$. Let $\mathbf{w}_1^{L+m}$ be the corresponding convolutional encoder output sequence. Assume that this convolutional encoder output sequence is transmitted over a discrete memoryless channel (A channel in which input and output are discrete and the output at a particular time instant depends only on the input at the same time instant). Let $\mathbf{s}_1^{L+m}$ be the corresponding channel output sequence. In this case, when $t < L$, there will be 2^k paths going out from each state in the trellis. When $t \leq m$, there will be only one path arriving at each state. On the other hand, when $t > M$, there will be 2^k paths arriving at each state. The MAP estimate of the encoder input sequence is given by

$$\hat{\mathbf{u}}_1^{*L+m} = \arg\max_{\mathbf{u}_1^{L+m}} P(\mathbf{u}_1^{L+m} | \mathbf{s}_1^{L+m}) \tag{A.1}$$

where $\mathbf{u}_t = (0, 0, \ldots, 0)$ when $t > L$. We know that

$$P(\mathbf{u}_1^{L+m} | \mathbf{s}_1^{L+m}) = \frac{P(\mathbf{u}_1^{L+m}, \mathbf{s}_1^{L+m})}{P(\mathbf{s}_1^{L+m})}.$$

Since the input-output mapping of a convolutional encoder is one-to-one, $P(\mathbf{u}_1^{L+m}, \mathbf{s}_1^{L+m}) = P(\mathbf{w}_1^{L+m}, \mathbf{s}_1^{L+m})$, and $P(\mathbf{s}_1^{L+m})$ does not affect the result, (A.1) is equivalent to maximizing

$$\begin{aligned} \log P(\mathbf{w}_1^{L+m}, \mathbf{s}_1^{L+m}) &= \log P(\mathbf{s}_1^{L+m} | \mathbf{w}_1^{L+m}) + \log P(\mathbf{w}_1^{L+m}) \\ &= \log P(\mathbf{s}_1^{L+m} | \mathbf{w}_1^{L+m}) + \log P(\mathbf{u}_1^{L+m}) \\ &= \sum_{t=1}^{L+m} \log P(\mathbf{s}_t | \mathbf{w}_t) + \log P(\mathbf{u}_1^{L+m}) \end{aligned} \tag{A.2}$$

The use of logarithm does not affect the maximization since it is a monotonic function. To simplify the term $\log P(\mathbf{u}_1^{L+m})$ further, it is required to know the statistical distribution of the information sequence $\mathbf{u}_1^L$. Let's consider the case where $\mathbf{u}_t$ is i.i.d when $t \leq L$. In this case,

$$\log P(\mathbf{u}_1^{L+m}) = \sum_{t=1}^{L+m} \log P(\mathbf{u}_t)$$

where $P(\mathbf{u}_t = (0, 0, \ldots, 0)) = 1$ when $t > L$. Therefore, (A.2) can be written as

$$\log P(\mathbf{w}_1^{L+m}, \mathbf{s}_1^{L+m}) = \sum_{t=1}^{L+m} \log P(\mathbf{s}_t | \mathbf{w}_t) + \log P(\mathbf{u}_t). \tag{A.3}$$

The term $\log P(\mathbf{w}_1^{L+m}, \mathbf{s}_1^{L+m})$ is called the metric associated with the path (in the trellis) or the path metric for the codeword $\mathbf{w}_1^{L+m}$. The term $\log P(\mathbf{s}_t | \mathbf{w}_t) + \log P(\mathbf{u}_t)$ in the summation in (A.3) is associated with a branch in the trellis and therefore called as branch metric. The summation of the metrics of the first t branches of a path is called as partial path metric (PPM) of the path segment up to time t. Now, the VA can be used to search the trellis for the path with the largest path metric. The VA involves finding the path with the largest PPM arriving at each state in the trellis. This path is called as the survivor for the corresponding state.

The Viterbi Algorithm

Step 1 At time $t = m$ find the single path (hence the survivor) arriving at each state and calculate the corresponding partial path metric. Store these paths with their partial path metrics.

Step 2 Increase t by 1. For each state, find the PPM of each of the arriving paths by adding the corresponding metric of the branch entering the state to the PPM of the connected survivor at the previous time unit. For each state, find the path with the largest PPM, store it with its PPM and eliminate all the other paths.

Step 3 If $t < m + L$ go to Step 2. Otherwise stop.

At time $t = L + m$, there is only one state in the trellis and hence one survivor. It can be shown [19] that this survivor is the path with the largest metric. Therefore, in this case, the VA results in the MAP estimate of the encoder input sequence.

Reference

[1] N. S. Jayant, "Subsampling of a DPCM speech channel to provide two self-contained half-rate channels," *Bell System Technical Journal*, vol. 60, no. 4, pp. 501–509, April 1981.

[2] A. A. El Gamal and T.M. Cover, "Achievable rates for multiple descriptions," *IEEE Transaction on Information Theory*, vol. 28, pp. 851–857, November 1982.

[3] L. Ozarow, "On a source coding problem with two channels and three receivers," *Bell System Technical Journal*, vol. 59, no. 10, pp. 1909–1921, December 1980.

[4] V. K. Goyal, "Multiple description coding: compression meets the network," *IEEE Signal Processing Magazine,* vol. 18, no. 5, pp. 74–93, September 2001.

[5] V. A. Vaishampayan,"Design of multiple description scalar quantizers," *IEEE Transactions on Information Theory,* vol. 39, pp. 821–834, May 1993.

[6] Y. Wang, M. T. Orchard, and A. R. Reibman, "Multiple description image coding for noisy channels by pairing transform coefficients," in *Proceedings of IEEE Workshop on Multimedia Signal Processing*, pp. 419–424, June 1997.

[7] V. K. Goyal, J. Kovacevic, and M. Vetterli, "Multiple description transform coding: Robustness to erasures using tight frame expansions," in *Proceedings IEEE International Symposium on Information Theory*, p. 408, August 1998.

[8] Y. Linde, A. Buzo and R. Gray, "An algorithm for vector quantizer design," *IEEE Transactions on Communications*, vol. COM-28, pp. 84–95, January 1980.

[9] A. Gersho, and R. Gray, *Vector quantization and signal compression.* Kluwer Academic Publishers, Norwell, MA., 1992.

[10] P. Koulgi, S. L. Regunathan, and K. Rose, "Multiple description quantization by deterministic annealing," *IEEE Transactions on Information Theory*, vol. 49, no. 8, pp. 2067–2075, August 2003.

[11] P. Yahampath, "On index assignment and the design of multiple description quantizers," in *proceedings of ICASSP* 2004, pp. 597–600, May 2004.

[12] K. Sayood and J. C. Borkenhagen, "Use of residual redundancy in the design of joint source channel coders," *IEEE Transactions on Communications*, vol. 39, pp. 839–846, June 1991.

[13] K. Sayood, F. Liu, and J. D.Gibson, "A constrained joint source/channel coder design," *IEEE Transactions on Communications*, vol. 12, pp. 1584–1593, December 1994.

[14] P. Yahamapath and U. Samarawickrama, "Joint source-channel decoding of convolutionally encoded multiple descriptions," accepted for publication in *Proceedings of the IEEE GLOBECOM'05*, November 2005.

[15] M. Sirinavasan,"Iterative decoding of multiple descriptions," in *Proceedings of Data Compression Conference DCC'99*, pp. 463–472, September 1999.

[16] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near Shannon limit error-correcting coding and decoding: Turbo-codes," in *Proceedings of ICC'93*, pp. 1064–1070, May 1993.

[17] J. Barrõs, J. Hagenauer, and N. Görtz, "Turbo cross decoding of multiple descriptions," in *Proceedings of IEEE ICC,* vol. 3, pp. 1398–1402, May 2002.

[18] G. D. Forney, Jr.,"The Viterbi algorithm," *Proceedings of the IEEE,* vol. 61, pp. 268–278, March 1973.

[19] S. Lin and D. J. Costello, *Error Control Coding (2nd edition)*, Prentice-Hall, Upper Saddle River, N.J., 2004.

[20] L. R. Bahl, J. Cocke, F. Jelinek and J. Raviv,"Optimum decoding of linear codes for minimizing symbol error rate," *IEEE Transactions on Information Theory,* vol. IT-20, pp. 284–287, September 1974.

[21] E. Paaske, "Short binary convolutional codes with maximal free distance for rates 2/3 and 3/4," *IEEE Transactions on Information Theory,* vol. IT-20, pp. 683–689, September 1974.

[22] N. Seshadri and C.-E. W. Sundberg, "List Viterbi decoding algorithms with applications," *IEEE Transactions on Communications,* vol. 42, pp. 313–323, Feb./Mar./Apr. 1994.

[23] U. Samarawickrama and P. Yahampath, "Joint-source channel decoding of multiple descriptions," in *IEEE Proceedings of the CCECE 2005*, May 2005.

[24] J. Hagenauer and P. Höher. A Viterbi Algorithm with Soft- Decision Outputs and its Applications. In *Proceedings of the IEEE GLOBECOM'89,* pp. 47.1.1–47.1.7, November 1989.

[25] C. Nill and C.-E. W. Sundberg, "List and soft symbol output Viterbi algorithms: Extensions and comparisons," *IEEE Transactions on Communications,* vol. 43, pp. 277-287, Feb./Mar./Apr. 1995.

[26] B. Chen and C.-E. W. Sundberg, "List Viterbi algorithms for continuous transmission," *IEEE Transactions on Communications,* vol. 49, no. 5, pp. 784–792, May 2001.

[27] J. B. Cain, G. C. Clark, and J. M. Geist, "Punctured convolutional codes of rate $(n-1)/n$ and simplified maximum likelihood decoding," *IEEE Transactions on Information Theory,* vol. IT-25, pp. 97–100, January 1979.

[28] N. Farvardin, "A study of vector quantization for noisy channels," *IEEE Transactions on Information Theory,* vol. 36, pp. 799–809, July 1990.