PROBABILISTIC ALGORITHM FOR LIST VITERBI DECODING

by

Janani Kalyanam

A thesis submitted in partial fulfillment

of the requirements for the degree of

Master of Science

(Electrical and Computer Engineering)

at the

University of Wisconsin - Madison

2009

Abstract

Digital communication has seen tremendous growth in the past two decades. A digital communication system can be broken down into the following components: channel encoder, communication channel, and channel decoder. The encoder adds structured redundancy into the data, the communication channel introduces noise into the system while the decoder performs algorithms to estimate the original data from a noisy one. There exist numerous encoding and decoding algorithms, each of which is characterized by various parameters like accuracy, memory, computational complexity and domain of applicability.

Convolutional codes are a class of channel codes that are used in a variety of applications like CDMA, GSM-cellular, deep space communication, 802.11 wireless LANs etc. The widely used decoding algorithm for convolutional codes is the Viterbi algorithm. A Viterbi algorithm produces a maximum-likelihood estimate of the original data. A list-Viterbi algorithm (of list size L) produces L best estimates of the original data. Although results show an improvement in performance when a list-Viterbi decoder is used as opposed to a Viterbi decoder, implementing a list-Viterbi decoder can prove to be expensive in terms of memory and computations.

In this thesis, we propose an novel probabilistic algorithm to emulate a list-Viterbi decoder. The motivation of the algorithm stems from the enormous complexity and memory requirements involved in a list-Viterbi algorithm, and the inexistence of an off-the-shelf list-Viterbi module. Our probabilistic algorithm uses an off-the-shelf Viterbi algorithm module, and using prior information, 'pins' the noisy sequence a number of times to recreate the list. We present a complexity comparison of the probabilistic algorithm against the list-Viterbi algorithm. We conclude the thesis by addressing some of the issues that arise while applying the algoithm to convolutional codes.

Acknowledgements

I am indebted to my advisor, Professor Stark Draper for his guidance and encouragement. His commitment and drive to the highest standards of research and his dedication to the vocation of mentorship have been inspiring. His simultaneous keen attention to minute technical details and broad overview has enormously helped shape my research. I have learned greatly from the several discussions with him, and hope to inherit his style of approach to research and problem solving.

This work was funded in part by Donald and Esther Procknow fellowship.

To my father on his 58^{th} birthday

Contents

List of Figures

PROBABILISTIC ALGORITHM FOR LIST VITERBI DECODING

Under the supervision of Professor Stark Draper At the University of Wisconsin, Madison

Digital communication has seen tremendous growth in the past two decades. A digital communication system can be broken down into the following components: channel encoder, communication channel, and channel decoder. The encoder adds structured redundancy into the data, the communication channel introduces noise into the system while the decoder performs algorithms to estimate the original data from a noisy one. There exist numerous encoding and decoding algorithms, each of which is characterized by various parameters like accuracy, memory, computational complexity and domain of applicability.

Convolutional codes are a class of channel codes that are used in a variety of applications like CDMA, GSM-cellular, deep space communication, 802.11 wireless LANs etc. The widely used decoding algorithm for convolutional codes is the Viterbi algorithm. A Viterbi algorithm produces a maximum-likelihood estimate of the original data. A list-Viterbi algorithm (of list size L) produces L best estimates of the original data. Although results show an improvement in performance when a list-Viterbi decoder is used as opposed to a Viterbi decoder, implementing a list-Viterbi decoder can prove to be expensive in terms of memory and computations.

In this thesis, we propose an novel probabilistic algorithm to emulate a list-Viterbi decoder. The motivation of the algorithm stems from the enormous complexity and memory requirements involved in a list-Viterbi algorithm, and the inexistence of an off-the-shelf list-Viterbi module. Our probabilistic algorithm uses an off-the-shelf Viterbi algorithm module, and using prior information, 'pins' the noisy sequence a number of times to recreate the list. We present a complexity comparison of the probabilistic algorithm against the list-Viterbi algorithm. A sligt modification of the probabilistic algorithm is also discussed. We con-

clude the thesis by addressing some of the issues that arise while applying the algoithm to convolutional codes.

Approved:

Professor Stark Draper

Department of Electrical and Computer Engineering

University of Wisconsin - Madison

Chapter 1

Introduction

1.1 Motivation

We shall make an attempt to answer the question - what does data communication entail? The essence of data communication is the ability to send messages from source to destination. A communication system can be abstracted into following blocks.

- source encoder and decoder: Source encoder uses various schemes to compress the incoming data and represents it with fewer number of bits. Source decoder performs the reverse action of data compression.

- channel encoder and decoder: Channel encoder introduces structured redundancy to the incoming information sequence to protect against errors while the decoder uses schemes to extract the original information from the received, noisy sequence.

- communication channel: is the physical medium through which information is transmitted. Typical parameters that characterize a channel include channel capacity, bandwidth etc.

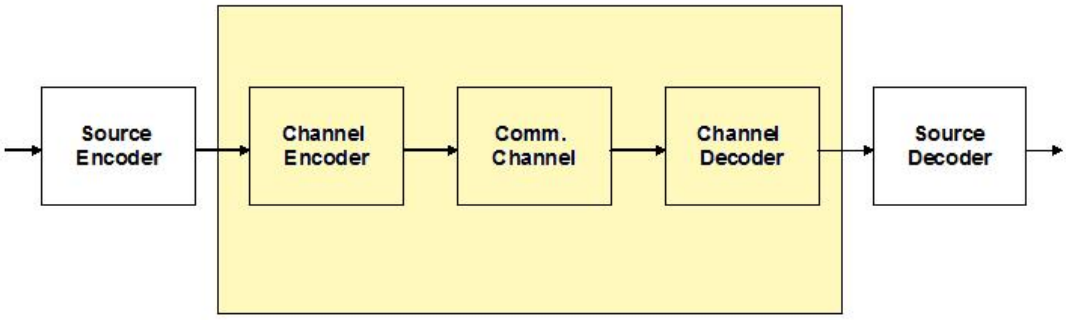

Figure 1.1: High-level architecture of a digital communication system

The focus of this thesis shall be the inner cover of fig[1.1] involving the channel encoder/decoder, and the communication channel. Channel capacity refers to the maximum amount information that can be transmitted through the channel. The channel coding theorem requires the rate of the channel code (which is a measure of the amount of redundancy added to the information sequence) to be less than channel capacity to ensure reliable transmission of information. It has been shown that rates of random codes of long lengths achieve capacity in the limit. However, due the absence of 'structure' in the addition of redundancy, the decoding scheme is very in-efficient. In fact, one may have to perform brute-force decoding of searching through the entire codebook. Hence, it is important that there is structure in the redundancy. There exist various algorithms that perform structured encoding, which greatly reduce the complexity of the corresponding decoding algorithms.

In this thesis, we have presented a novel and efficient probabilistic decoding algorithm for convolutional codes, which is shown to out-perform the existing list decoding algorithm, the list-Viterbi algorithm. The received, noisy information sequence is 'pinned' at various indices and fed into a maximum-likelihood decoder each time. The output produced from the maximum-likelihood decoder is used to populate a list of code word estimates. We analyze the performance of the algorithm for randomly generated codebook, and show results comparing the probabilistic algorithm with list-Viterbi algorithm.

1.2 Thesis Outline

The rest of the thesis is organized in the following way. We start Chapter 2 by briefing introductions to channel coding, and list decoding. The rest of Chapter 2 is spent discussing convolutional codes, Viterbi algorithm and list-Viterbi algorithm. The complexity of the list-Viterbi algorithm is also derived to be $O(L\log(L))$, where L is the list size.

In Chapter 3, we motivate the need for an alternate algorithms for list-Viterbi decoding and present a probabilistic algorithm to perform list-Viterbi decoding. We formulate the problem of probabilistic list-decoding for any maximum-likelihood decoder, and derive the trade off between probability of error and algorithm complexity.

In Chapter 4, we address some issues faced while applying the algorithm to convolutional codes. We end the thesis with some concluding remarks in Chapter 5.

Chapter 2

List Decoding and Viterbi algorithm

2.1 Channel Coding

We work with the abstracted version of a digital communication system that consists of channel encoder, communication channel, and channel decoder. The encoder adds structured redundancy to a length-k binary information sequence $\mathbf{u}$, and produces a length-n code word $\mathbf{v}$ which is sent through a channel. The channel introduces noise into the system. We will consider a discrete memoryless binary symmetric channel (BSC-p) which is parametrized by the cross over probability, p. The decoder sees a noisy received sequence $\mathbf{r}$, and exploits the structure in the redundancy added at the encoder to produce an estimate of the code word $\hat{\mathbf{v}}$ (and equivalently an estimate of the corresponding information sequence, $\hat{\mathbf{u}}$) that was sent.

While designing a communication system, one is faced with the task of providing a cost effective system at a certain level of reliability that is acceptable to the user. Usually reliability is measured in terms of probability of error, $P(\mathbf{v} \neq \hat{\mathbf{v}})$. Since the use of channel coding techniques involve adding redundancy to the information bits, it will not only demand for an increased bandwidth, but will also result in an increased system complexity. The system complexity arises from implementing efficient algorithms in the decoder to retrieve

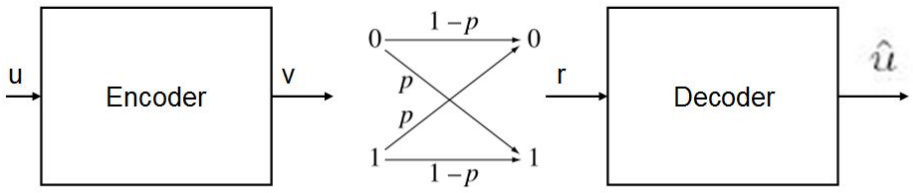

Figure 2.1: Communication System

the original information. Hence, bandwidth and system complexity are two things that need to be considered before employing a coding scheme.

2.2 Maximum Likelihood Estimate

We formulate the following problem: X and Y are binary random variables. Assume that X is Bernoulli($\frac{1}{2}$). Y is related to X by the following equation: $X = Y \oplus N$, where N is a Bernoulli(p). We observe Y, and would like to find $\hat{X}$, the 'best' estimate of X. There exist many fidelity criterions which can define 'best'. The Minimum Mean Square Estimate (MMSE) corresponds to $\hat{X} = \underset{X}{\text{argmin}}\ E[(X - \hat{X})^2]$. We shall consider the Maximum Likelihood Estimate (MLE) criterion, $\hat{X} = \underset{X}{\text{argmax}}\ P(X|Y)$.

$$\hat{X} = \underset{X}{\text{argmax}}\ P(X|Y) \tag{2.1}$$

$$= \underset{X}{\text{argmax}}\ \frac{P(Y|X)P(X)}{P(Y)} \tag{2.2}$$

$$= \underset{X}{\text{argmax}}\ P(Y|X) \tag{2.3}$$

Since X and Y are both Bernoulli($\frac{1}{2}$) random variables, we omit $P(X)$ and $P(Y)$ from eq[2.2]. Note that we can think of X and Y as being separated by a BSC(p). $P(Y|X)$

is equal to the cross-over probability of the BSC. If $p < \frac{1}{2}$, $\hat{X} = \underset{X}{\operatorname{argmin}}\, d_H(X, Y)$, where d_H stands for the hamming distance between X and Y. Therefore, for a BSC the MLE corresponds to the code word word which has least hamming distance with respect to the received sequence. We shall focus on ML decoder for the purpose of this work.

2.3 List Decoding

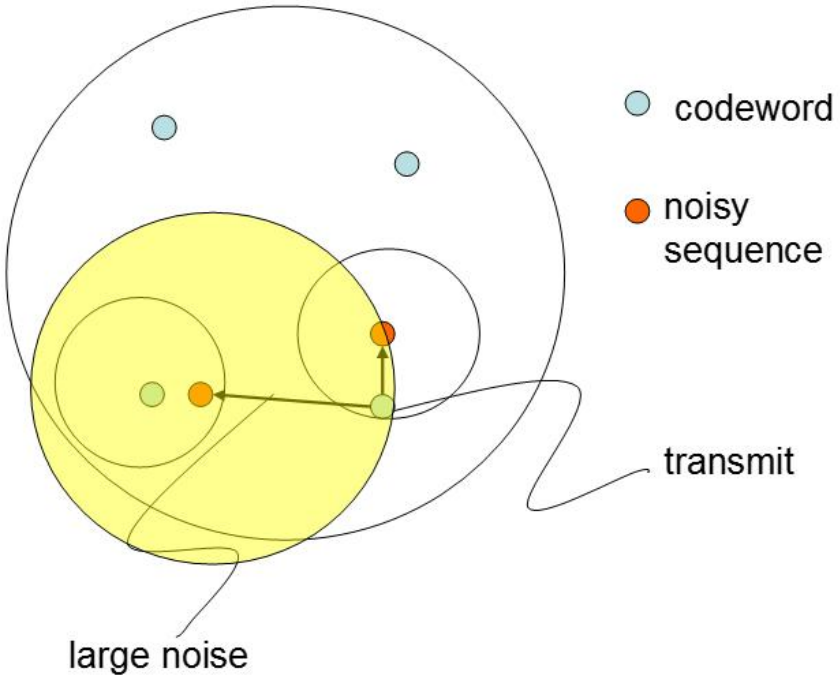

Figure 2.2: Depicts space of code words

In the system we considered so far, the decoder produces a single best guess of the information sequence. We shall now take a look at what happens when noise is large. In Figure 2.2, we see that when the noise is large the noisy sequence gets thrown far away from the original code word. And a minimum hamming distance decoding would not yield the correct answer. This brings us to the notion of list decoding. In Figure 2.2, the decoder is asked to produce a list 2 MLEs of the code word. i.e. it produces the closest and the second

closest code word with respect to the received sequence. And we see that the original code words is indeed present in the list. In list decoding an error is declared only when the code word is not present in the list. For a length-n code word, the list size L usually satisfies $L << n$. A reason for impressing a bound on L is that, if L were large the decoding problem becomes the trivial case of listing all the code words in the codebook [1]. If L is small, then the set of L most likely code words form $\mathcal{L}$.

$$\mathcal{L} = \begin{pmatrix} \leftarrow \mathbf{v}_1 \rightarrow \\ \leftarrow \mathbf{v}_2 \rightarrow \\ \leftarrow \mathbf{v}_3 \rightarrow \\ \cdots \\ \leftarrow \mathbf{v}_L \rightarrow \end{pmatrix}$$

2.4 Convolutional Codes

There exist two major types of error correcting codes: block codes, and convolutional codes. The main difference between the two lie in the fact that there is memory in convolutional codes, where as are block codes are memoryless. Convolutional codes are generated from FIR filters. A convolutional code is parametrized by (n, k, K), where $\frac{k}{n}$ is the rate of the code, and K is referred to as the constraint length. In general, a convolutional encoder consists of K (k bit) shift registers. Entries in the registers are combined in a linear fashion to produce n output bits [3].

At each time instant, the entries from the last K shift registers are dropped, and all the previous entries are shifted by K registers. When a new batch of input bits enter the encoder, the encoder can be in one of $2^{(K-1)k}$ possible states. Since the output of the encoder at any particular time not only depends on the current input, but also on the current state of the

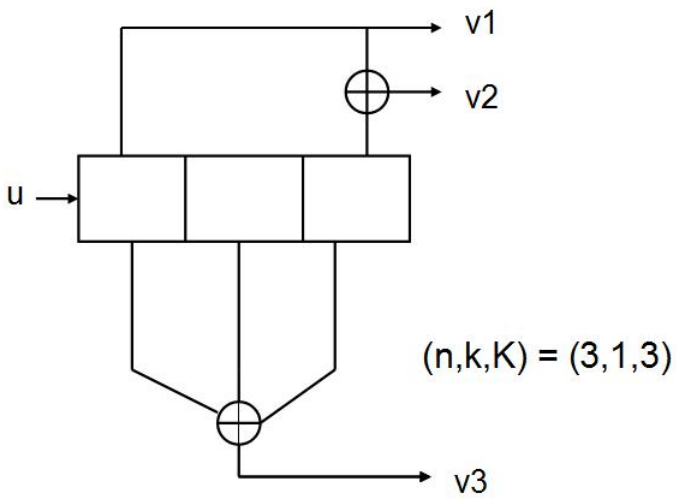

Figure 2.3: Rate $\frac{1}{3}$ convolutional encoder. Constraint length = 3. When a new bit comes in, the state of the encoder is $00, 01, 10$ or 11

shift registers, such an encoder can be thought of as a finite state machine, more precisely a mealy machine. Another important representation of the encoder is the trellis diagram. The vertical axis in a trellis diagram (fig[2.4]) represents the possible states of the encoder, and the horizontal axis represents time evolution. The trellis diagram is in essence a finite state machine evolving in time. The picture of a trellis is an important one to keep in mind since it easens the process of describing the decoding scheme for a convolutional encoder.

Figure[2.4] describes the trellis structure of a rate $\frac{1}{3}$ code. The four states in the trellis correspond to states $00, 01, 10$ and 11. The dashed lines represent the state transition due to input 1 and the solid lines represent state transitions due to input bit 0. In the next section, we describe Viterbi algorithm.

2.5 Viterbi Algorithm

The Viterbi algorithm comes up often in many applications that use Hidden Markov Models. In Natural Language Processing, the purpose of Parts of Speech (POS) tagging is to

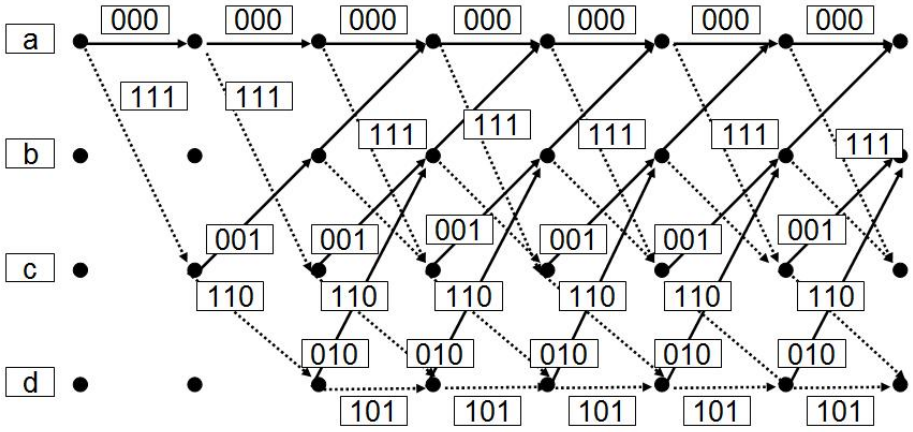

Figure 2.4: Rate $\frac{1}{3}$ convolutional encoder viewed in trellis form

label each word of a sentence with its POS label. Given an input (word) sequence $w_{1:N}$, we want to produce a sequence of labels $\hat{x}_{1:N}$ such that $\hat{x}_{1:N} = \underset{x_{1:N}}{\operatorname{argmax}}\, p(x_{1:N}|w_{1:N})$. The Viterbi algorithm performs efficient maximum likelihood decoding on noise-affected convolutionally encoded sequences. Given the trellis structure of the encoder and the observed noisy sequence, the Viterbi algorithm finds the best path through the trellis. As mentioned in section[2.2], finding the MLE for BSC reduces to finding the code word with least hamming distance with respect to the received sequence. We shall provide a summary of the Viterbi algorithm here. We use similar notations as in Seshadri and Sundber [5].

We consider M sections of a fully connected N state trellis. There are a total of N^2M paths in the trellis. We shall represent the incremental cost of transition from state j at time $t-1$ to state i at time t as $c_t(j,i)$. The algorithm boils down to finding the path of minimum cost starting from, say state 1 and ending at state 1. The best path to state j at time t is given by $\phi_t(j)$. Let the history of the best path be stored in $\xi_t(j)$.

1. Initialization: $(t = 1)$

$$\begin{aligned} \phi_1(i) &= c_1(1, i) \\ \xi_1(i) &= 1 \\ 1 &\leq i \leq N \end{aligned}$$

2. At time t:

$$\begin{aligned} \phi_t(i) &= \min_{1 \leq j \leq N} [\phi_{t-1}(j) + c_t(j, i)] \\ \xi_t(i) &= \operatorname*{argmin}_{1 \leq j \leq N} [\phi_{t-1}(j) + c_t(j, 1)] \\ 1 &\leq i \leq N \end{aligned}$$

3. Termination:

$$\begin{aligned} \phi_M(i) &= \min_{1 \leq j \leq N} [\phi_{M-1}(j) + c_M(j, i)] \\ \xi_M(i) &= \operatorname*{argmin}_{1 \leq j \leq N} [\phi_{M-1}(j) + c_M(j, 1)] \end{aligned}$$

4. Traceback: The sequence with minimum cost corresponds to

$$(1, i_1, \ldots, i_{M-1}, 1),$$

where

$$i_t = \xi_{t+1}(i_{t+1}),$$

$$1 \leq t \leq M - 1$$

2.6 List Viterbi Algorithm

$\phi_t(i,k)$ shall represent the k^{th} lowest cost to reach state i at time t. $\xi_t(i,k)$ shall be the state (and $r_t(i,k)$ be the corresponding ranking) of the k^{th} best path at time $t-1$ when this path passes through state i at time t.

1. Initialization: $(t = 1)$

$$\begin{aligned} \phi_1(i,k) &= c_1(1,i) \\ \xi_1(i) &= 1 \\ 1 &\leq i \leq N \\ 1 &\leq k \leq L \end{aligned}$$

2. At time t:

$$\begin{aligned} \phi_M(i,k) &= \min^k_{\substack{1\leq j\leq N \\ 1\leq l\leq L}} [\phi_{M-1}(j,l) + c_t(j,i)] \\ (j^*,i^*) &= \operatorname*{argmin}^k_{\substack{1\leq j\leq N \\ 1\leq l\leq L}} [\phi_{M-1}(j,l) + c_t(j,i)] \\ 1 &\leq i \leq N \end{aligned}$$

Here $\min^k$ is the k^{th} smallest value, j^* is the predecessor of the k^{th} best path into state i and l^* is the corresponding ranking.

$$\begin{aligned} \xi_t(i,k) &= j^* \\ \gamma_{i,k} &= l^* \end{aligned}$$

3. Termination:

$$\begin{aligned}
\phi_M(i,k) &= \min_{\substack{1\le j\le N\\ 1\le l\le L}}{}^{k}[\phi_{M-1}(j,l)+c_M(j,1)] \\
(j^*,i^*) &= \operatorname*{argmin}_{\substack{1\le j\le N\\ 1\le l\le L}}{}^{k}[\phi_{M-1}(j,l)+c_M(j,1)] \\
\xi_t(i,k) &= j^* \\
\gamma_{i,k} &= l^*
\end{aligned}$$

4. Traceback: The sequence with k^{th} minimum cost corresponds to

$$(1, j_1, \ldots, j_{M-1}, 1),$$

where

$$j_t = \xi_{t+1}(j_{t+1}, l_{t+1}),$$

$$l_t = r_{t+1}(j_{t+1}, l_{t+1})$$

$$l_{M-1} = r_M(i,k)$$

At each stage the algorithm keeps track of the L best paths in the trellis. Hence, there is a sort that happens at each stage to find the L paths with least cost. Hence, the complexity of the list Viterbi algorithm with respect to the list size L is $O(L \log L)$, detailed proof of which is presented in appendix A.

Chapter 3

Probabilistic Algorithm

3.1 Introduction

In Chapter 2, we described list-Viterbi algorithm and studied its complexity with respect to list size L to be $O(L \log L)$. In this chapter, we present an alternate probabilistic algorithm for list-Viterbi decoding. We formulate the problem on a random codebook generated from i.i.d Bernoulli(q) distribution. Hence, code words in the codebook are pairwise independent and bitwise independent. The received sequence is pinned to a '0' or a '1' at various indices each time and the modified received sequence is fed into an ML decoder. The pinning is repeated M times. The output of the ML decoder populates a guess-list (GL) of code words, $\hat{\mathcal{L}}$. An expression for probability that the original list (OL) of code words, $\mathcal{L}$, is a subset of GL, $Pr(\mathcal{L} \subseteq \hat{\mathcal{L}})$ is derived. We find an approximate expression for M in terms of a given probability of error, ϵ and codebook parameter q.

3.2 Terminology

Some new terminology is introduced for this problem. We also revise some concepts that were introduced in Chapter 2 and put them into the context of this problem.

3.2.1 ML Decoder

In Chapter 2, we introduced MLE to be $\hat{X} = \underset{X}{\text{argmax}}\ P(X|Y)$. It was also proved (for Bernoulli($\frac{1}{2}$) variables X and Y) that MLE for a BSC boils down to finding the code word closest to the received sequence, where 'closest' is in terms of hamming distance. A naive way of finding the MLE would be to calculate the hamming distance between every code word and the received sequence, and to pick the one which achieves minimum hamming distance. If mulitple code words achieve minimum hamming distance, one of them is chosen randomly. This process would require calculating the hamming distance of 2^{nR} code words, where n is the length of the code word, and R is the rate of the code. The Viterbi algorithm avoids this exponential amount of calculation by picking the most likely path at each stage, and there by eliminating the any descendants of the other less-likely paths. In this problem, we will consider an ideal ML decoder which will spit out the closest code word to the received sequence as the MLE.

3.2.2 'Pinning'

The received sequence is 'pinned' to 0 or 1 at a randomly chosen index. 'Pinning' an index sets infinite confidence on observing that index to be a particular bit. For example, if the i^{th} bit of the received sequence $\mathbf{r}$ is pinned to be 0, and the modified $\mathbf{r}$ is fed into the ML decoder, we can be assured that the i^{th} bit of the output of the decoder will be 0. If an index has been pinned, the ML decoder searches only over the subset of code words which are 0 at the i^{th} bit. Hence, the resulting code word is closest to the modified received sequence.

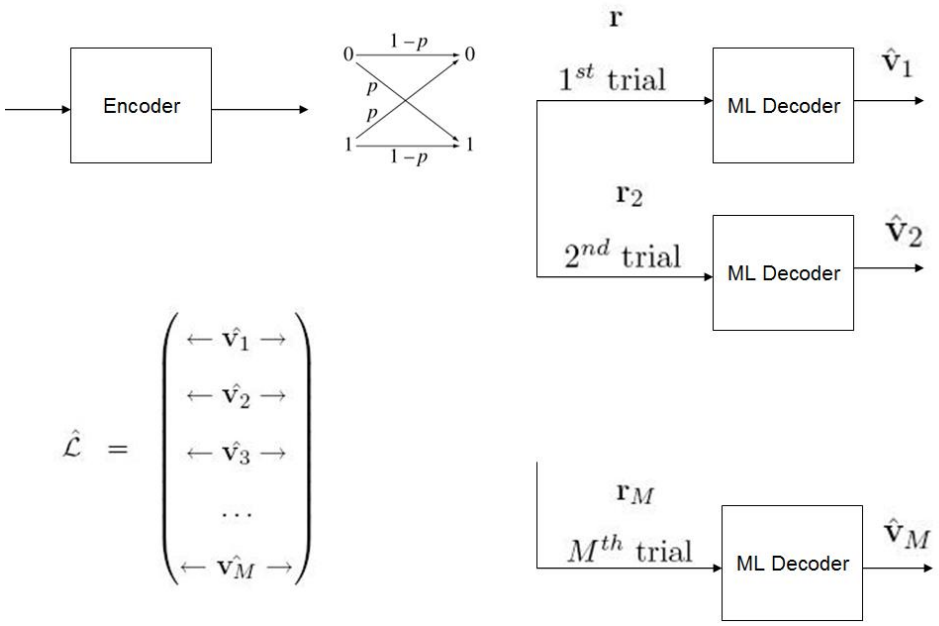

Figure 3.1: Block diagram of system implementing probabilistic algorithm. The output of the ML decoder at each iteration will populate $\hat{\mathcal{L}}$

3.2.3 α parameter

In solving for $Pr(\mathcal{L} \subseteq \hat{\mathcal{L}})$, a randomly generated codebook is considered. That is, each code word is generated by independent flips of a Bernoulli(q) distribution. The probability of generating a particular code word is $q^h(1-q)^{n-h}$ where h is the number of 1s in the code word. Given a particular index, probability that any two code words differ in that index is given by $2q(1-q)$. Probability that the two code words match at a given index is $q^2+(1-q)^2$, or $1-2q(1-q)$. We define $\alpha := 2q(1-q)$ and $\beta := 1-\alpha$

3.3 Algorithm Description

A length-n noisy sequence $\mathbf{r}$ is observed at the decoder. On the first trial, $\mathbf{r}$ is sent through an ML decoder. The resulting code word $\hat{\mathbf{v}}_1$ will be the first entry in GL, $\hat{\mathcal{L}}$. Note that $\hat{\mathbf{v}}_1$ will also be the first entry of OL, $\mathcal{L}$ since it is the MLE. Set $\mathcal{I}$ is initialized to be the set

of positive integers between 1 and n. On the second trial, an element i from $\mathcal{I}$ is chosen at random. We set $\mathbf{r}_2 = \mathbf{r}$ except the i^{th} bit is set to be $\mathbf{r}_2(i) = \overline{\hat{\mathbf{v}}_1(i)}$. The modified received sequence $\mathbf{r}_2$ matches $\mathbf{r}$ at all indices except at the i^{th} index, where it may or may not match. $\hat{\mathbf{v}}_2$ becomes the second entry in $\hat{\mathcal{L}}$. $\mathcal{I}$ is pruned to be the subset of indices where the entries of $\hat{\mathcal{L}}$ match. During the third trial, a similar procedure is followed. An element i from $\mathcal{I}$ is chosen at random. The following assignments are made:

$$\mathbf{r}_3 = \mathbf{r} \tag{3.1}$$

$$\mathbf{r}_3(i) = \overline{\hat{\mathbf{v}}_1(i)} \tag{3.2}$$

$\mathbf{r}_3$ is fed into the ML decoder and the resulting code word $\hat{\mathbf{v}}_3$ becomes the third entry of $\hat{\mathcal{L}}$. This procedure is repeated for a total of M times, the first being the case when $\mathbf{r}$ is directly fed into the ML decoder without any modification. The algorithm can be summarized in the following steps:

1. find ML estimate $\hat{\mathbf{v}}_1$
2. $\mathcal{L} = (1, 2, \ldots n)$
   (a) randomly choose $i \in \mathcal{I}$
   (b) look up ML estimate, flip bit and **pin** on $\mathbf{r}$
   (c) feed modified received sequence to ML algorithm
   (d) prune $\mathcal{I}$
   (e) jump to step (a)

3.4 Comments on the algorithm:

Some important observations about the algorithm are discussed below:

- The OL $\mathcal{L}$ contains distinct code words. Hence, in the probabilistic algorithm, we want to utilize M trials to produce distinct code word outputs to populate GL. Note that in Eq[3.2], the i^{th} bit is not only set differently from the ML code word, but is set differently from all entries so far of $\hat{\mathcal{L}}$. Although not explicit from the equation, this is true because $\mathcal{I}$ pruned to be the set of indices where the entries so far of $\hat{\mathcal{L}}$ match. Hence this algorithm is guaranteed to produce distinct code word outputs.

- At every step, $\mathbf{r}$ is modified and fed into the ML decoder. It should be noted that, in order for one of the M guesses to be the second most likely code word, $\mathbf{v}_2$, all it takes is: the randomly chosen index i at some iteration t has to be an index where $\mathbf{v}_2$ and $\mathbf{v}_1$ don't match. If that is true, by arguments made in sections 3.2.1 and 3.2.2 we can be assured that the resulting guess $\hat{\mathbf{v}}_t = \mathbf{v}_2$. Same arguement holds true for every code word in $\hat{L}$ except of course $\mathbf{v}_1$ since it is the ML estimate.

- This algorithm works well for random codes. To intuitively understand this proposition, the probability with which the second most likely code word is obtained in the second guess itself is considered. For a codebook generated from Bernoulli($\frac{1}{2}$), any two code words differ in about $\frac{n}{2}$ indices. In particular, this holds true when the two code words happen to be $\mathbf{v}_1$, and $\mathbf{v}_2$. And in sections 3.2.1 and 3.2.2 it was argued if i (in any of the M iterations) happens to be an index where $\mathbf{v}_1$ and $\mathbf{v}_2$ differ, then with 100% surity, the output of the ML decoder will indeed be the second-most-likely code word. Hence, $P(\mathbf{v}_2 \in \hat{\mathcal{L}}) = \frac{1}{2}$. Similarly, $P(\mathbf{v}_3 \in \hat{\mathcal{L}}) = \frac{1}{4}$ and so on.

- The set $\mathcal{I}$ is pruned at every iteration to be the subset of indices where the entries so far of $\hat{\mathcal{L}}$ match. It is possible that if $\hat{\mathbf{v}}_1$ and $\hat{\mathbf{v}}_2$ happen to be bit-wise complements of each other, the algorithm runs out of indices in the third iteration itself. In the problem formulation, it is assumed that n is large, so the probability of $\hat{\mathbf{v}}_1$ and $\hat{\mathbf{v}}_2$ being bits-wise complements of each other is negligible.

- $M \geq L$ has to be true. Else $P(\mathcal{L} \subseteq \hat{\mathcal{L}}) = 0$. That is, we at least need L trials to recreate a list of size L.

We want to find the $P(\mathcal{L} \subseteq \hat{\mathcal{L}})$ as a function of M. The result, proof of which is provided in the appendix, is stated as a theorem below.

Theorem 3.4.1. *For a random codebook generated from Bernoulli(q), the probability that $\mathcal{L} \subseteq \hat{\mathcal{L}}$ is given by*

$$P(\mathcal{L} \subseteq \hat{\mathcal{L}}) = \prod_{i=1}^{L-1}(1 - (1-\alpha)^{M-i}) \tag{3.3}$$

where $\alpha = 2q(1-q)$, and M is the number of times the received sequence is pinned.

3.5 ϵ, M tradeoff

In the previous section, probability that the original list will be recreated was derived. In this section, we provide results expressing M in terms of probability of error, which is $P(\mathcal{L} \nsubseteq \hat{\mathcal{L}})$, ϵ.

$$\begin{aligned} P(\mathcal{L} \subseteq \hat{\mathcal{L}}) &= 1-\epsilon \\ \prod_{i=1}^{L-1}(1 - (1-\alpha)^{M-i}) &= 1-\epsilon \\ \sum_{i=1}^{L-1} \log\left(1 - (1-\alpha)^{(M-i)}\right) &= \log(1-\epsilon) \end{aligned}$$

Using the notation $\beta = 1 - \alpha$ and simplifying, we arrive at the following equation.

$$M = \frac{\log(-\log(1-\epsilon)) - \log\left(\frac{1}{\beta}\right) - \log\left(\frac{1-\left(\frac{1}{\beta}\right)^{L-1}}{1-\frac{1}{\beta}}\right)}{\log\beta} \tag{3.4}$$

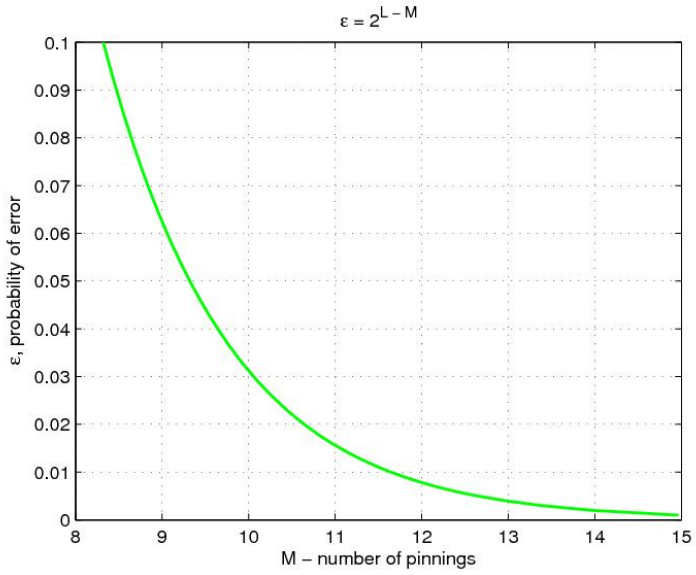

Figure 3.2: Enumerates the decay of prob. of error, ϵ as a function of M for list size, $L = 5$. $\alpha = \beta = \frac{1}{2}$

Using Taylor's approximation, and substituting $\beta = \frac{1}{2}$,

$$\epsilon \approx 2^{L-M} \tag{3.5}$$

It can be seen from eq[3.5] that the probability of error drops exponentially as M grows.

- Figure[3.2] and figure[3.3] show ϵ as a function of M for $\alpha = \frac{1}{2}$.
- Figure[3.4] shows the comparison of complexity of list-Viterbi ($L \log L$) against the probabilistic algorithm.

3.6 Proof sketch for Thm 3.4.1

Events A_i for $2 \leq i \leq L$ are defined to be events that the i^{th} most likely code word is present in $\hat{\mathcal{L}}$. Hence, $P(\mathcal{L} \subseteq \hat{\mathcal{L}})$ is the joint probability of events $A_2, A_3 \ldots A_L$. In Eq[3.3], the RHS is a product of $L-1$ terms. Each term of the $L-1$ terms corresponds to the $P(A_i)$ for $2 \leq i \leq L$

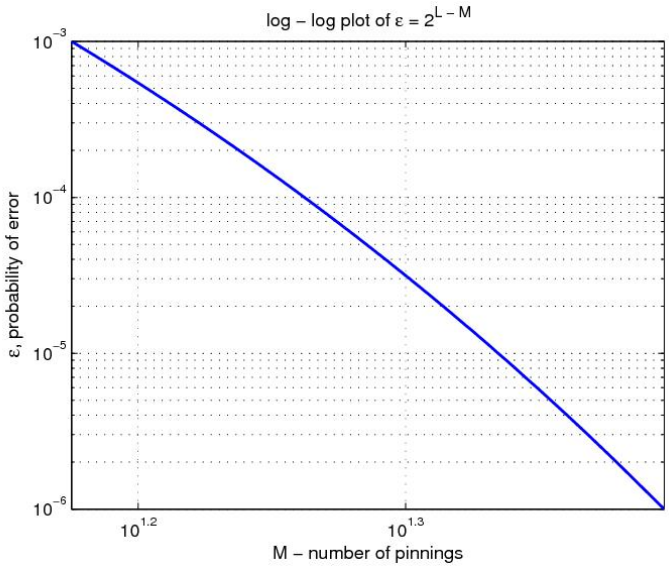

Figure 3.3: log-log plot of ϵ as a function of M for list size, $L = 5$. $\alpha = \beta = \frac{1}{2}$

respectively. To intuitively understand these terms, we consider $P(A_2^c) = (1-\alpha)^{M-1}$. In section 2.3 and 4, it was explained that α corresponds to the probability that an index is 'favorable' to a particular code word. Hence $(1-\alpha)$ is the probability that $\hat{\mathbf{v}}_2 \neq \mathbf{v}_2$. Since the pinning index is chosen independently in each trial, $(1-\alpha)^{M-1}$ is the probability that $\mathbf{v}_2$ is never on the GL, $\hat{\mathcal{L}}$.

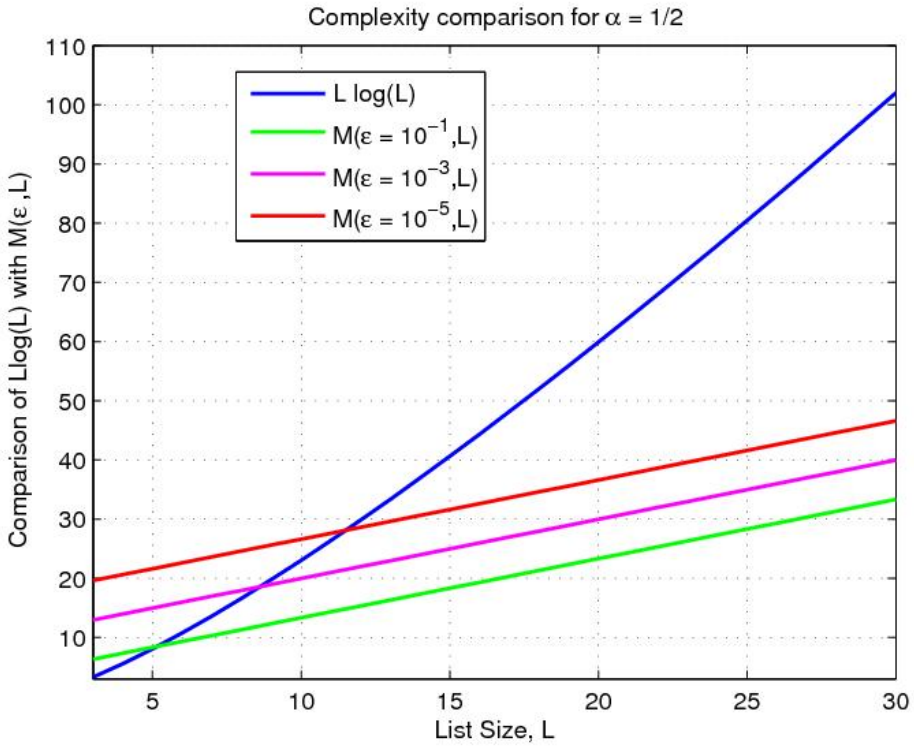

Figure 3.4: Comparison of complexity of list-Viterbi algorithm against probabilistic for various ϵ values

Chapter 4

Application to Convolutional Codes

4.1 Introduction

In chapter 3, we described a probabilistic algorithm to emulate a list ML decoder. In this chapter, we discuss some of the issues involved in applying such an algorithm to convolutional codes, and list-Viterbi decoding.

4.2 Applying to convolutional codes

In this section of the chapter, we discuss some of the issues that arise while applying the algorithm to convolutional codes.

4.2.1 Codebook structure

The probabilistic algorithms works well for random codes. 'Pinning' indices are randomly chosen from a set of legitimate indices. The larger the percentage of 'favorable' indices, greater is the probability of recreating the list. 'Favorable' indices to a certain code word correspond to the indices in which it differs from $\mathbf{v}_1$. If the codebook is generated from a

Bernoulli($\frac{1}{2}$) distribution, then there is a 50% chance for each bit to be 'favorable' for $\mathbf{v}_2$. (Since the codebook is randomly generated, the code words are bit-wise independent.)

In the case of convolutional codes, the code words are not randomly generated. There is lot of structure in the code words. Moreover, since the encoder is a finite impulse response filter, the code words are not bit wise independent either. Hence, there exists a very small subset of indices which are favorable for each code word. Consequently, probability of choosing the correct bit reduces greatly.

4.2.2 α parameter

The role of α parameter is studied in context of section 4.2.1. α is the parameter of the codebook which determines how random the codebook actually is. Setting $\alpha = \frac{1}{2}$ would result in a very random codebook, as opposed to a more conservative value of α, for example, 0.1 which would produce a more structured codebook as in the case of convolutional codes. This idea suggests a greater value of M for a particular probability of error, ϵ for a greater α.

4.2.3 Subset of legitimate indices

At every iteration, the subset of legitimate indices is the subset of indices where the entries so far of $\hat{\mathcal{L}}$ match. On an average, any two code words in the codebook match in about $n(1-\alpha)$ bits. Hence, at iteration t, the cardinality of the subset of legitimate indices decreases as $n(1-\alpha)^t$. When $\alpha = \frac{1}{2}$, any two code words, on an average differ on about $\frac{n}{2}$ bits. However, it is also possible that our second guess differs from the $\hat{\mathbf{v}}_1$ in all the indices. This would make the subset of legitimate indices a null set in the second iteration itself. This did not show up in our end result because of our assumption that n is large. But this is an important issue that needs to be kept in mind while implementing the algorithm.

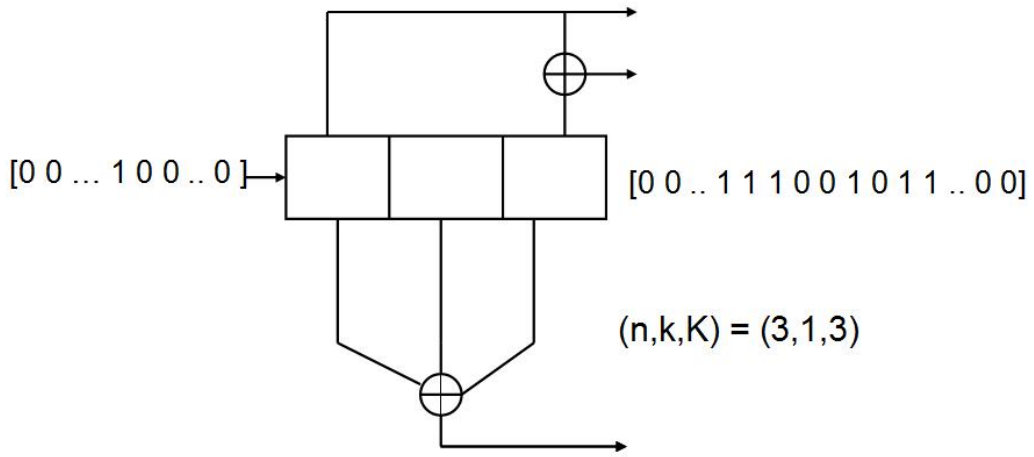

Figure 4.1: Convolutional encoder of a Rate $\frac{1}{3}$ convolutional code. Depending on where the impulse occurs at the input, the finite response is seen in the code word. Hence, there exist many code words with the same sequence of bits (111001011) placed different positions.

4.2.4 Example of Rate $\frac{1}{3}$ code

To understand the consequences of the issues better, rate $\frac{1}{3}$ convolutional code is considered. Since convolutional codes are linear codes, and the spectrum of possible errors for any code words is the same as the all zeros code word, it suffices just to consider the all zeros code word sent to the decoder over a BSC. Suppose that the most likely code word, $\mathbf{v}_1$, is indeed decoded to be the all zeros code word. We want to develop some insight into what the subsequent entries of the the original list, $\mathcal{L}$ might be. In other words, we want to intuitively understand what $\mathbf{v}_2$, $\mathbf{v}_3$ etc might be.

The impulse response of the encoder (FIR filter) is a code word of weight 6, meaning there are 6 1s in the code word. We remind ourselves about the encoder of a rate $\frac{1}{3}$ convolutional code. In fig[4.2.4], the sequence of 6 bits can be placed anywhere in the code word depending where the spike occurs in the information sequence. Hence, any one of these code words would form a legitimate second most likely code word. Hence, the original list is not unique.

Hence, to recreate the list, it suffices if the list hamming distances of every code word in $\mathcal{L}$ to $\mathbf{r}$ matches the list of hamming distances of the top L candidates in $\hat{\mathcal{L}}$. The top candidates of $\hat{\mathcal{L}}$ are the code words from $\hat{\mathcal{L}}$ that have the L least hamming distances to the received sequence. This is precisely the reason why it suffices to have the condition $\mathcal{L} \subseteq \hat{\mathcal{L}}$ satisfied instead of $\mathcal{L} = \hat{\mathcal{L}}$.

Chapter 5

Summary and Conclusion

A list Viterbi algorithm produces a ranked list of L best candidates after a trellis search. We showed that the complexity of a list Viterbi algorithm with respect to list size is $O(L\log(L))$. In this thesis, we have proposed an alternate probabilistic algorithm to reconstruct the original list with lesser complexity which is given by M. There exist many contexts where list decoding is relevant. In this thesis, list decoding was motivated as a solution to the scenario where the noise is high. It was argued that, the received sequence (in the presence of high noise levels) will be thrown far away from the original code word that the decoder ceases to decode to the correct code word. On the other hand, if a list of outputs are produced by the decoder, then an error occurs only when the original code word is not present in the list.

A list decoder may be useful in circumstances where the transmitted data contains redundancy, or when a feedback channel is available. If a feedback channel is available, the transmitter could be made aware of the entries of the decoder's list either by direct transmission or by feeding back what was actually received and letting the transmitter know what the decoder's scheme would be. In this case, the transmitter would have to send only the additional information to the decoder to resolve the issue of choosing one among the L

estimates [1]. A list decoder is also useful if there is internal information to the decoder. For example, in transmission of English letters, any incorrect code word will look like garble and can hence be discarded from the list [1].

Chapter 4 discusses some important issues that come up while applying the probabilistic list decoding scheme to a Viterbi decoder (or convolutional codes). The analysis for probabilistic list decoding assumes a random code book, and an ideal ML decoder which would output the closest codeword to the received sequence. Although Viterbi decoder is indeed an ML decoder, convolutional codes have a lot of structure in them. Hence the random pinning aspect which worked greatly for a random code book would fail for convolutional codes. It is our stance that having more information on where which indices to pin would greatly help apply the algorithm to convolutional codes.

Another issue that was discussed in Chapter 4 was that, the subset of legitimate pinning indices decreases approximately exponentially with each iteration. We are currently exploring an alternate algorithm which doesn't maintain a subset of legitimate pinning indices, but allows the pinning at each iteration to be completely random. While this overcomes the problem an exponentially decreasing legitimate pinning index set, it allows for the guesses to be repeated. It is our intuition that, given M tries, a majority of the tries would be spent in decoding to the second most likely sequence, and the next major portion of the effort with be spent in decoding to the third most likely sequence and so on.

Appendix

Appendix A

Complexity of list Viterbi

We analyze the complexity of the list Viterbi algorithm with respect to

- k - the length of the information sequence
- S - number of states
- L - list size

The algorithm can be broken down into,

- At each time instant,
  - at each state
    - calculate S new metrics, S operations
    - add each of the above metrics to a length L vector, LS operations
    - sort a vector of length LS, complexity $O(LS \log LS)$
    - calculate rank, involves loop of size L, hence complexity $O(L)$
  - trace back to produce sequence of state transitions, loop size k
  - trace forward through the state transitions to produce an estimate-sequence, loop size k

Hence, the expression for complexity is

$$f(k,S,L) = kS\left[S + LS + LS\log LS + L\right] + 2k$$

Complexity w.r.t S

$$\begin{aligned} f(k,S,L) &= kS\left[S + LS + LS\log LS + L\right] + 2k \\ &= kS^2 + kLS^2 + kLS^2\log LS + kSL + 2k \\ &\approx O(S^2) + O(S^2) + O(S^2\log S) + O(S) \\ &\approx O(S^2\log S) \end{aligned}$$

Complexity w.r.t k

$$\begin{aligned} f(k,S,L) &= kS\left[S + LS + LS\log LS + L\right] + 2k \\ S &= 2^{(K-1)k}, \text{where K is the constraint length.} \\ f &= k * 2^{(K-1)k}\left[2^{(K-1)k} + L * 2^{(K-1)k}\log\left(L * 2^{(K-1)k}\right) + L\right] + 2k \\ \text{say } \xi &= 2^{(K-1)} \\ f &= k\xi^{2k} + k\xi^{2k}L + k\xi^{2k}L\left(\log L + k\log\xi\right) + k\xi^{k}L + 2k \\ &= O(k\xi^{2k}) + O(k\xi^{2k}) + O(k\xi^{2k}) + O(k^2\xi^{2k}) + O(k\xi^{k}) + O(k) \\ &\approx O(k^2\xi^{2k}) \end{aligned}$$

Complexity w.r.t L

$$\begin{aligned} f(k,S,L) &= kS\left[S + LS + LS\log LS + L\right] + 2k \\ &= O(L) + O(L\log L + O(L) \\ &\approx O(L\log L) \end{aligned}$$

Appendix B

Pinning Algorithm derivation

We prove Thm[3.4.1] here.

Proof. We use v_i to denote the i^{th} most likely codeword, and $\hat{v}_i$ to denote the i^{th} guess out of the probabilistic algorithm.

$$\mathcal{L} = \begin{pmatrix} \leftarrow v_1 \rightarrow \\ \leftarrow v_2 \rightarrow \\ \leftarrow v_3 \rightarrow \\ \cdots \\ \leftarrow v_L \rightarrow \end{pmatrix}$$

$$\hat{\mathcal{L}} \;=\; \begin{pmatrix} \leftarrow \hat{v}_1 \rightarrow \\ \leftarrow \hat{v}_2 \rightarrow \\ \leftarrow \hat{v}_3 \rightarrow \\ \cdots \\ \leftarrow \hat{v}_M \rightarrow \end{pmatrix}$$

We define A_i, for $2 \leq i \leq L$, as the event that the i^{th} most likely codeword is in $\hat{\mathcal{L}}$. We also note that the probability that any two codewords differ in index k is given by $2q(1-q)$. Therefore, if the pinning index happens to be an index where v_1 and v_j differ, the PA will produce v_j as its output (provided $v_2, v_3, \ldots v_{j-1} \in \hat{\mathcal{L}}$.)

$$P(\mathcal{L} \subseteq \hat{\mathcal{L}}) \;=\; P(A_2, A_3 \ldots A_L)$$

Using Bayes' Rule

$$\begin{aligned}
P(A_2, A_3 \ldots A_L) &= P(A_2)P(A_3|A_2)\ldots P(A_L|A_2^{L-1}) \\
P(A_2) &= 1 - P(A_2^c) \\
&= 1 - P(v_2 \notin \hat{\mathcal{L}}) \\
P(A_2^c) &= P(\bigcap_{i=2}^{M} v_2 \neq \hat{v}_i) \\
&= P(v_2 \neq \hat{v}_2)P(v_2 \neq \hat{v}_3|v_2 \neq \hat{v}_2)\ldots P(v_2 \neq \hat{v}_M|v_2 \neq \hat{v}_2, v_2 \neq \hat{v}_3, \ldots v_2 \neq \hat{v}_{M-1}) \\
&= (1-\alpha)^{M-1} \\
P(A_2) &= 1 - (1-\alpha)^{M-1}
\end{aligned}$$

Now, we derive an expression for $P(A_3|A_2)$.

$$P(A_3|A_2) = 1 - P(A_3^c|A_2) \quad \text{(B.1)}$$

$$= \frac{P(A_3^c \cap A_2)}{P(A_2)} \quad \text{(B.2)}$$

The numerator of Eq(B.2) becomes

$$\begin{aligned} P(A_3^c \cap A_2) &= P((A_3^c \cap (v_2 = \hat{v}_2)) \cup (A_3^c \cap (v_2 = \hat{v}_3)) \cup \ldots \cup (A_3^c \cap (v_2 = \hat{v}_M))) \\ &= \sum_{i=2}^{M} P(A_3^c \cap (v_2 = \hat{v}_i)) \\ &= \sum_{i=2}^{M} P(v_2 = \hat{v}_i) P(A_3^c|(v_2 = \hat{v}_i)) \\ P(A_3^c|(v_2 = \hat{v}_i)) &= P(\bigcap_{j=2}^{M} (v_3 \neq \hat{v}_j)|(v_2 = \hat{v}_i)) \\ &= P(v_3 \neq \hat{v}_2|v_2 = \hat{v}_i) \ldots P(v_3 \neq \hat{v}_M|v_3 \neq \hat{v}_2, v_3 \neq \hat{v}_3, \ldots v_3 \neq \hat{v}_{M-1}, v_2 = \hat{v}_i) \end{aligned}$$

In the above equation, there are $M-1$ terms, out of which one of the terms is 1. Each of the other terms is $(1 - 2q(1-q))$. So, we have

$$\sum_{i=2}^{M} P(v_2 = \hat{v}_i) P(A_3^c|(v_2 = \hat{v}_i)) = (1 - 2q(1-q))^{M-2} \sum_{i=2}^{M} P(v_2 = \hat{v}_i)$$

Using law of total probability,

$$\begin{aligned} P(v_2 = \hat{v}_i) = \quad & P(v_2 = \hat{v}_i|\bigcap_{j=2}^{i-1}(v_2 \neq \hat{v}_j)) P(\bigcap_{j=2}^{i-1}(v_2 \neq \hat{v}_j)) \\ & + P(v_2 = \hat{v}_i|\bigcup_{j=2}^{i-1}(v_2 = \hat{v}_j)) P(\bigcup_{j=2}^{i-1}(v_2 = \hat{v}_j)) \end{aligned}$$

Note that the second term in the above expression is 0 and the first term is just

$$2q(1-q)(1-2q(1-q))^{i-2}.$$

Hence we have that,

$$\begin{aligned} P(v_2 = \hat{v}_i) &= 2q(1-q)(1-2q(1-q))^{(i-2)} \\ &= \alpha(1-\alpha)^{(i-2)} \\ \sum_{i=2}^{M} P(v_2 = \hat{v}_i) &= \alpha \sum_{i=2}^{M} (1-\alpha)^{(i-2)} \\ &= 1-(1-\alpha)^{(M-1)} \end{aligned}$$

Hence, the numerator of Eq(B.2) becomes,

$$\begin{aligned} \frac{P(A_3^c \cap A_2)}{P(A_2)} &= \frac{(1-\alpha)^{M-2}(1-(1-\alpha)^{M-1})}{(1-(1-\alpha)^{M-1})} \\ P(A_3^c|A_2) &= (1-\alpha)^{M-2} \\ P(A_3|A_2) &= 1-(1-\alpha)^{M-2} \end{aligned}$$

Similarly, we can show that $P(A_4|A_2, A_3) = 1-(1-\alpha)^{M-3}$ and so on. So, we have

$$\begin{aligned} P(\mathcal{L} \subseteq \hat{\mathcal{L}}) &= P(A_2, A_3 \ldots A_L) \\ &= \prod_{i=1}^{L-1} (1-(1-\alpha)^{M-i}) \end{aligned}$$

□

Appendix C

Approximation

We simplify the expression $\prod_{i=1}^{L-1}(1-(1-\alpha)^{M-i})$, and obtain ϵ as a function of M.

We want

$$\sum_{i=1}^{L-1} \log\left(1-(1-\alpha)^{(M-i)}\right) = \log\left(1-\epsilon\right)$$

Say $\beta = 1-\alpha$. Now,

$$\sum_{i=1}^{L-1} \log\left(1-\beta^{M-i}\right) = \log(1-\epsilon)$$

Simplifying LHS,

$$\sum_{i=1}^{L-1} \log\left(1-\beta^{M-i}\right)$$

$$\begin{aligned}
&\approx \sum_{i=1}^{L-1} -\beta^{M-i} \\
&= -\beta^{M} \sum_{i=1}^{L-1} \beta^{-i} \\
&= -\beta^{M} \left(\frac{1}{\beta}\right) \left(\sum_{j=0}^{L-2} (\frac{1}{\beta})^{j}\right) \\
&= -\beta^{M} \left(\frac{1}{\beta}\right) \left(\frac{1-\left(\frac{1}{\beta}\right)^{L-1}}{1-\frac{1}{\beta}}\right)
\end{aligned}$$

Now, we have

$$\begin{aligned}
-\beta^{M} \left(\frac{1}{\beta}\right) \left(\frac{1-\left(\frac{1}{\beta}\right)^{L-1}}{1-\frac{1}{\beta}}\right) &\approx \log\left(1-\epsilon\right) \\
M &\approx \frac{\log\left(-\log\left(1-\epsilon\right)\right) - \log\left(\frac{1}{\beta}\right) - \log\left(\frac{1-\left(\frac{1}{\beta}\right)^{L-1}}{1-\frac{1}{\beta}}\right)}{\log\beta}
\end{aligned}$$

We subsitute $\beta = \frac{1}{2}$ and simplify the terms in the above equation,

$$\begin{aligned}
\log\left(\frac{1-\left(\frac{1}{\beta}\right)^{L-1}}{1-\frac{1}{\beta}}\right) &= \log\left(\frac{1-2^{L-1}}{1-2}\right) \\
&= \log 2^{L-1} - 1 \approx L-1 \\
\log\frac{1}{\beta} &= -\log\beta = 1
\end{aligned}$$

Hence,

$$\begin{aligned} M &\approx \frac{\log\left(-\log\left(1-\epsilon\right)\right)-\left(L-1\right)-1}{-1} \\ &\approx L-\log\left(-\log\left(1-\epsilon\right)\right) \end{aligned}$$

Solving for ϵ we get

$$\epsilon \approx 1-2^{\left(-2^{L-M}\right)}$$

By Taylor's approximation, $\log\left(1-\epsilon\right) \approx -\epsilon$. From the above equation, we have

$$\begin{aligned} \epsilon &\approx 1-\left(\frac{1}{2}\right)^{\left(\frac{1}{2}\right)^{M-L}} \\ 1-\epsilon &\approx \left(\frac{1}{2}\right)^{\left(\frac{1}{2}\right)^{M-L}} \\ \log\left(1-\epsilon\right) &\approx \log\left(\frac{1}{2}\right)^{\left(\frac{1}{2}\right)^{M-L}} \\ -\epsilon &\approx -\left(\frac{1}{2}\right)^{M-L} \\ \epsilon &\approx 2^{L-M} \end{aligned}$$

Bibliography

[1] G. D. Forney, *Exponential Error Bounds for Erasure, List, and Decision Feedback Schemes*, IEEE Transactions of Information Theory, vol 14, pp 206-220, March 1968

[2] E.A. Lee and D.G. Messerschmitt, *Digital Communication*, Kluwer Academic Publishing, Norwell, MA, 1996.

[3] J.G. Proakis, *Digital Communications*, McGraw-Hill, New York, NY, 2001.

[4] S.C. Draper and E. Martinian, *Compound conditional source coding, Slepian-Wolf list decoding, and applications to media coding*, in proceedings of International Symposium of Information Theory, July 2007.

[5] N. Seshadri and C.W. Sundberg, *List Viterbi Decoding Algorithms with Applications*, IEEE Transactions on Information Theory, vol 42, pp 313-323, February/March/April 1994.