ECE 545 - Communication Theory
Spring 2004

Instructor: Dr. Predrag Spasojevic

Final Project Report

Maximum Likelihood Sequence Detection with Convolutional Encoding in Presence of Inter-Symbol Interference (ISI) and AWGN

By
Baik Hoh
Neville Clemens
Shankar Somasundaram
Shweta Shrivastava

Abstract *– This report performs the performance analysis and simulation of a Maximum Likelihood Sequence Estimator in presence of Inter-Symbol Interference and Additive White Gaussian Noise (AWGN) in the channel. Comparison is done for such a system with and without convolutional encoding. The receiver uses a Viterbi decoder.*

1. Introduction

Inter-symbol interference (ISI) arises in pulse amplitude modulated systems whenever the effects of one transmitted pulse are not allowed to die away completely before the transmission of next. ISI may also occur in signals that propagate over multipath channels, over cables with dispersive characteristics, or via filtering circuits. ISI is a major impediment to reliable communication. The Viterbi decoding algorithm provides a way of equalizing the effects of ISI in channels.**[FN2]** The Viterbi decoder is a Maximum Likelihood Sequence Estimator (MLSE).**[FN1]** This paper describes a mechanism to use the Viterbi Equalizer for a channel with known finite memory and performs a comparison of such equalization with and without convolutional encoding in the system. The paper also explains a derivation of upper and lower bounds on the probability of error for such a channel.

2. Analysis

For our analysis, **[OM]** we consider the following system.

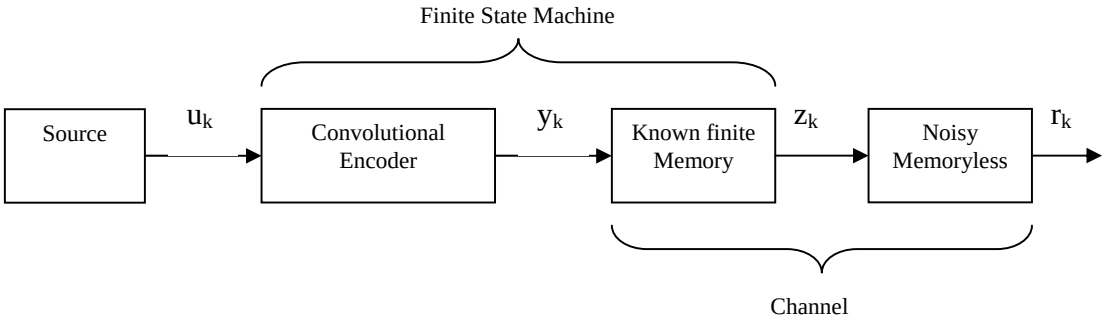

Figure 1. General Block Diagram of the System

General Part: We assume that the source generates a sequence, $\{u_k\}$ of equally likely statistically independent bits at a rate of R bits per second. These information bits are fed to a convolutional encoder of memory v_1. The encoder output at time k is y_k and is given by

$$y_k = f(k, u_{k-1}, u_{k-2}, ..., u_{k-v_1}) \text{ ------ (1)}$$

The encoder output is fed to the finite memory part of the channel, whose output at time k is given by

$$z_k = g(k, y_k, y_{k-1}, ..., y_{k-v_2-1}) \text{ ------ (2),}$$

where v_2 is the memory of the channel.

The above equations suggest that z_k is directly dependent on the input sequence and hence we can write,

$$z_k = h(k, u_{k-1}, u_{k-2}, ..., u_{k-v}) \text{ ------ (3),}$$

where $v = v_1 + v_2 - 1$ is the total memory of the concatenation of convolutional encoder and finite memory part of the channel. These two components can be considered to form a finite state machine, whose state $\mathbf{x_k}$ is given by,

$$x_k = \begin{bmatrix} u_{k-1} \\ . \\ . \\ u_{k-v} \end{bmatrix} \text{------ (4)}$$

Suppose we consider a finite information sequence $u_0, u_1, \ldots\ldots, u_{L-1}$, where $L >> v$. Assuming the initial state

$$x_0 = \mathbf{0} \text{ ------ (5)}$$

the receiver receives $r_1, r_2, \ldots r_L$ and must form an estimate sequence $\hat{u}_1, \hat{u}_2, ..., \hat{u}_{L-1}$ from all possible 2^L binary sequences where the criterion for the estimated sequence is minimum probability of error. The minimum probability of error sequence maximizes the conditional probability **[WJ]**

$$\Pr\{u_0, u_1, ..., u_{L-1} \mid r_1, r_2, ..., r_L\} \text{ ------ (6)}$$

By applying Baye's rule and condition for independence, our criteria reduces to

$$\Pr\{r_1, r_2, ..., r_L \mid u_0, u_1, ..., u_{L-1}\} \text{ ------ (7)}$$

At this point, we make the reasonable assumption that, given the initial state x_0, the output sequence of the finite state machine $z_1, z_2, ..., z_L$ uniquely determines the input sequence $u_0, u_1, ..., u_{L-1}$. This allows us to change the conditioning variables from $\{u_i\}$ to $\{z_i\}$. Also, considering the memoryless property of the channel, the criterion reduces to minimization of the cost function

$$J(z_1, z_2, ..., z_L) = -\sum_{k=1}^{L} \ln[\Pr\{r_k \mid z_k\}] \text{ ------ (8)}$$

We now assume that the last v information symbols are used to resynchronize the finite state machine to some known state so that the receiver knows the states $\hat{x}_0 = x_0$ and $\hat{x}_L = x_L$ exactly. Now suppose that an error occurs for the first time at state x_k, that is,

$$\begin{aligned} &\hat{u}_i = u_i; i = 0,1,..., k-1 \\ &\textit{and} \qquad\qquad \text{------ (9)} \\ &\hat{u}_k \neq u_k \end{aligned}$$

Clearly, we have,

$$\begin{aligned} &\hat{x}_i = x_i; i = 1,2,...,k \\ &\hat{x}_i \neq x_i; i = k+1, k+2,...,k+v \end{aligned} \quad \text{------ (10)}$$

For $i \geq k + v + 1$, the maximum likelihood states may or may not coincide ith the true states. Since, $\hat{x}_L = x_L$, there exists a time j in the interval $[k + v + 1, L]$ such that

$$\hat{x}_i \neq x_i; i = k+v+1, k+v+2,..., j-1 \quad \text{------ (11)}$$

and

$$\hat{x}_j \neq x_j \quad \text{------ (12)}$$

That is, there exists some time j in the interval from $k + v + 1$ to L where the maximum likelihood path merges with the true path for the first time after k. The following figure shows such a case.

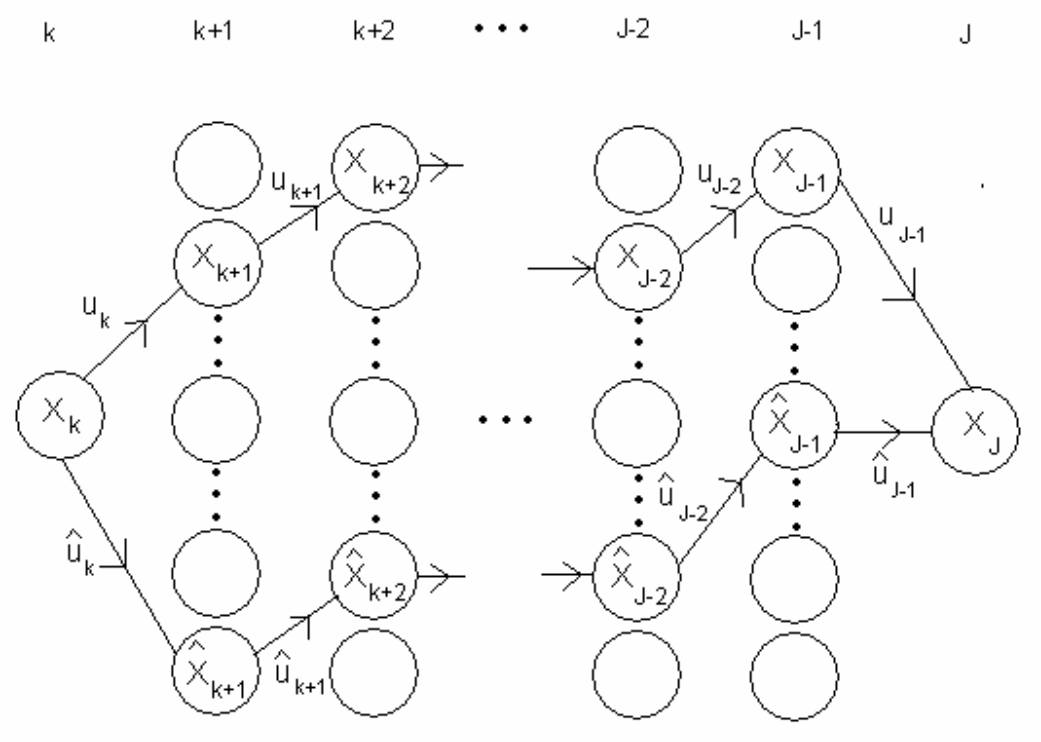

Fig2. Diagram of diverging paths.

Now, note that an error occurs for the first time at k if and only if some path diverges from the true path at x_k incurs the minimum cost

$$-\sum_{i=k+1}^{j-1} \ln[\Pr\{r_i \mid z_i\}] \quad \text{------ (13)}$$

while separated from the true path.

We can now make two definitions: **[PR]**

E_{kj} = the event that some path that diverges from the true path at x_k and merges back for the first time at x_j incurs less cost than any other path from x_k to x_j.

M_{kj} = the number of possible paths that diverge from the true path at x_k and merge with at x_j for the first time.

Clearly, M_{kj} is the number of information sequences that satisfy

$$\hat{x}_k = x_k, \hat{x}_i \neq x_i; i = k+1,..., j-1, \hat{x}_j = x_j \text{ ------ (14)}$$

and

$$M_{kj} = 2^{j-k-2-v} \text{ ------ (15.1)}$$

Clearly,

$$M_{kj} \leq 2^{j-k-v} \text{ ------ (15.1)}$$

The probability of error is thus given by,

$$P_e = \Pr\left\{\bigcup_{k=1}^{L-v} \bigcup_{j=k+v+1}^{L} E_{kj}\right\} \text{ ------ (16)}$$

which is upper bounded by the sum of the probability of each event:

$$P_e \leq \sum_{k=1}^{L-v} \sum_{j=k+v+1}^{L} \Pr\{E_{kj}\} \text{ ------ (17)}$$

We further upper bound the error probability by picking the allowable state sequence $x_k,...,x_j$ that minimizes $\Pr\{E_{kj}\}$. Hence,

$$P_e \leq \sum_{k=1}^{L-v} \sum_{j=k+v+1}^{L} \max_{x_k,...x_j} \Pr\{E_{kj}\} \text{ ------ (18)}$$

An even simpler bound is

$$P_e \leq L^2 \max_{k} \max_{j \geq k+v+1} \max_{xk,....xj} \Pr\{E_{kj}\} \text{ ------ (19)}$$

Recall that E_{kj} is the event that some path that diverges from the true path at x_k and merges back for the first time at x_j incurs less cost than any other path between the two states. There are M_{kj} possible paths that can do this. Again applying the union bound we get

$$\Pr\{E_{kj}\} \leq \sum_{i=1}^{M_{kj}} \Pr\{E_{kj}^i\} \text{ ------ (20)}$$

where E_{kj}^i is the path that diverges from the true path at x_k and merges back for the first time at x_j incurs less cost than the true path. We can evaluate the expression $\Pr\{E_{kj}^i\}$ for various target systems.

Specific Part: We can try to get $P\{E_{kj}^i\}$ for specific case in this project.

Target system description:
- ISI channel with additive white Gaussian noise (**ATTN:** look at how we model this channel at the **Appendix1**.)
- Viterbi decoder is used for sequence estimator.
- Convolutional encoding of information bits into multilevel symbols.

Based on our statement about error event, E_{kj}^i, it occurs if and only if

$$\Pr\{r_k, r_{k+1}, \ldots, r_j \mid z_k^i, z_{k+1}^i, \ldots, z_j^i\} \geq \Pr\{r_k, r_{k+1}, \ldots, r_j \mid z_k, z_{k+1}, \ldots, z_j\} \text{ ------ (21)}$$

Because the channel is memoryless, the joint pdf could be divided independently. If we take –ln(*) and rewriting, we can get the following result.

$$\sum_{l=k}^{j} (r_l - z_l^i)^2 \leq \sum_{l=k}^{j} (r_l - z_l)^2,$$
$$n > \frac{1}{2}\sqrt{\sum_{l=k}^{j} (z_l - z_l^i)^2} \quad \text{------ (22)}$$

E_{kj}^i is equivalent to a zero mean Gaussian random variable n with variance σ^2.

$$\text{Hence } \Pr\{E_{kj}^i\} = Q\left(\frac{d_{kj}^i}{2\sigma}\right) \text{ ------ (23),}$$

where $d_{kj}^i = \left[\sum_{l=k}^{j} (z_l - z_l^i)^2\right]^{1/2}$ is the distance between the two output sequences.

From now on, we have to find the range of d_{kj}^i. By referring to Omura's work **[OM]**, we can get the range using beautiful matrix notation like the following:

$$\beta_L{}^2 \sum_{l=k}^{j} (y_l - y_l^i)^2 \leq \sum_{l=k}^{j} (z_l - z_l{}^i)^2 \leq \beta_U{}^2 \sum_{l=k}^{j} (y_l - y_l^i)^2 \text{ ------ (24).}$$

If we use above inequality about d_{kj}^i and applying upper bound one more time ($Q(x) \leq e^{-\frac{x^2}{2}}$), we can get the upper bound for multilevel PAM system who has a convolutional coding with the channel in the presence of ISI.

$$P_e \leq L^2 \max_k \max_{j \geq k+v+1} \sum_{i=1}^{M_{kj}} \exp\left\{-\frac{\beta_L{}^2}{8\sigma^2} \sum_{l=k}^{j} (y_l - y_l{}^i)^2\right\}. \text{------(25)}$$

Rather than consider a specific convolutional encoder, suppose we consider an ensemble of encoders, **[WJ]** if we randomly select an encoder, it might give us the average error

probability. Definitely it is greater than the error probability of the best encoder. With this approach, we can get the upper bound for optimum encoder.

$$P_e(optimum) \le L^2 2^{-v_1 R_0} \quad \text{------ (26)}$$

where R0 is normally greater than 1 for moderate to large signal-to-noise ratios and multi-PAM signaling. This shows that we can find convolutional encoders such that the error probability can be made to decrease at least exponentially with v1, the memory of the convolutional encoder.

Notice!!!: *If you look at the equation (26), you can realize that it couldn't give an analysis graph which definitely should be compared to simulation graph. But, it alarms us that if we can use convolutional coding in transmitter, we can control the overall probability. Likewise the way we did in analysis, we can also simulate the system performance in the presence of ISI with varying the v1.*

A simple lower bound can be found by going back to our original expression for Pe.

$$\begin{aligned} P_e &= \Pr\left\{ \bigcup_{k=1}^{L-v} \bigcup_{j \ge k+v+1}^{L} E_{kj} \right\} \\ &\ge \max_{k} \max_{j \ge k+v+1} \max_{x_k,\ldots,x_j} \Pr\left\{E_{kj}\right\} \qquad \text{------ (27)} \\ &\ge \max_{k} \max_{j \ge k+v+1} \max_{x_k,\ldots,x_j} \max_{x_k^i,\ldots,x_j^i} \Pr\left\{E_{kj}^i\right\} \end{aligned}$$

Using our previous equations, a simple lower bound comes out.

$$P_e \ge Q\left(\frac{d^i_{\min}}{2\sigma}\right), \text{ where } d^i_{\min} = \min_{k} \min_{j \ge k+v+1} \min_{x_k,\ldots,x_j} \min_{x_k^i,\ldots,x_j^i} d^i_{kj} \text{ ------ (28)}$$

These lower bounds apply for a particular convolutional encoder but are independent of the transmitted information sequence. The upper bound on Probability error of optimum encoder is independent of the particular convolutional encoder since it represents an average over a class of encoders.

Thus, based on the result of analysis, we're going to simulate the following scheme.

- MLSD receiver in the presence of ISI.
- MLSD receiver with convolutional coding in the presence of ISI.

With the first scheme, we can verify how well MLSD cancels the ISI and how close it is to the BPSK system without ISI. After that, according to our result of analysis, we can get performance gain by varying convolutional coding memory length. From the simulation result, we can check our analysis result.

3. Simulation Environments and Assumptions

The following simulations were taken under some assumptions:

- We assume that receiver already knows about channel coefficients. **(See Appendix. (1))**
- For starting with zero state and ending with zero state, we append some zero padding bits to original information sequence. **(See Appendix. (1))**
- We use baseband equivalent discrete-time model for ISI channel which was already obtained using whitening filter **(See Appendix. (2))**
- We use dynamic programming for ML sequence estimation **[HY] (See also Appendix. (1))**

4. Simulation Algorithm and Results

The below graphs shows the performance of viterbi decoding for MLSD in the presence of ISI and white Gaussian noise and compares it to viterbi decoding used for the same channel without using Convolutional encoding.

(1) AWGN channel with channel coefficients [1 0.8 0.5]

The channel coefficients used for this simulation were [1 0.8 0.5] and the memory of the channel was 2.The convolution encoder used had a constraint length of 5 and the generator matrix in octal was 23,35.

Noise power was varied from -1 to -10 dB and since BPSK signaling was used, the signal energy was zero. Hence SNR=-Noise power and so SNR varied from 3 to dB.

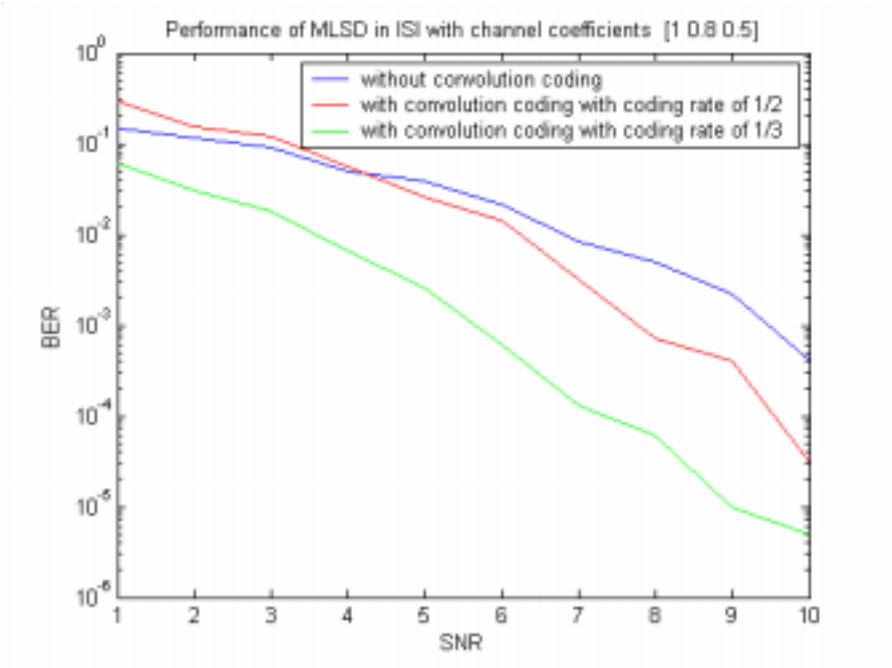

Figure 3. BER Performance I over ISI channel

As seen from the above graph, initially at 1-3 dB the performance without convolutional encoding was better as compared to the performance with convolutional encoding with rate 1/2. This is a trend observed at low SNR. But the difference between the 2 simulations as seen from the graph is very low.

At higher SNR's the performance of viterbi with convolution coding gets better and better .At around 10 dB, the Bit error rate without convolution coding was around 0.0004, but with convolution coding was around 0.00003, a power of 10 better.

With a convolution coding of rate 1/3 the performance is far better than with convolution coding with rate ½ or without convolution coding. This shows that as you increase the coding rate , the performance of the system in the presence of ISI and awgn gets better

(2) AWGN channel with exponentially decaying channel coefficients

The channel coeffcients generated for this simuation were generated from the equation f(k)=(sqrt(1-(a^2)))*(a^k) where a was selected as 0.5

The values of the channel coefficients were found to be [0.8660 0.4330 0.2165]

The other conditions remained the same as before. The graph below shows the performance with and without convolution for ISI with white Gaussian noise.

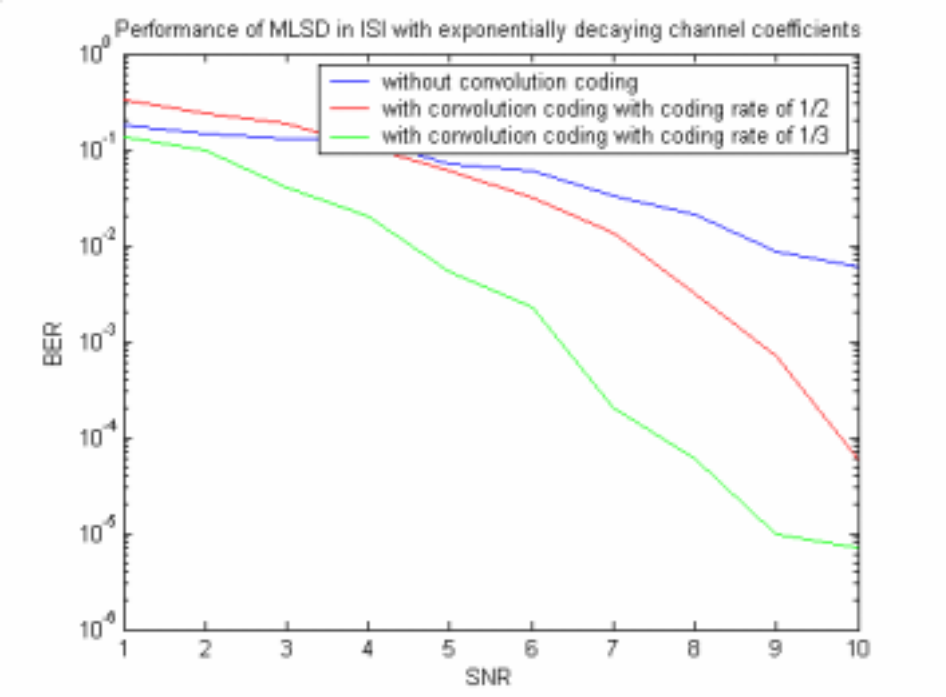

Figure 4. BER Performance II over ISI channel

As seen from the above graph, initially at lower SNR [1-3 dB] the performance without convolutional encoding was better as compared to the performance with convolutional encoding at rate 1/2. Also as seen before at higher SNR's the performance with convolution coding at rate 1/2 was found to be much better. As before the performance of convolution coding at higher coding rates gets better.

An important observation was that with exponentially decaying coefficients, the performance with and without convolution was worse as compared to the previous case. This was because in case of exponentially decaying channel coefficients, the dependence on the current bit was lower ie f_0 was 0.866 as compared to 1 for the previous case.

5. Conclusion

a. We prepared some required knowledge (Convolutional coding [PK], Viterbi algorithm [HY, FN1], ISI channel model with whitening filter [PK]) for analysis.

b. We derived the upper bound and lower bound of probability of error for the following system. (Target system: MLSD optimum receiver with convolutional coding in the presence of ISI and WGN)

c. According to result of analysis, we implemented simulator using MATLAB and took the required simulations. (Simulator has these items: convolutional coding, viterbi decoder and ISI channel model)

d. Based on the simulation results, we can get the BER performance gain by 1.5dB between with convolutional coding and without it for getting $P_e = 10^{-3}$

e. As we increase the memory length of convolutional encoder, we can get about 2dB performance gain for getting $P_e = 10^{-3}$ from coding rate=1/2 to 1/3

6. References

[FN1] G.D. Forney, Jr., "The Viterbi algorithm," Proc IEEE, vol.61, pp.268-278, Mar. 1973.

[HY] J.F. Hayes, "The Viterbi algorithm applied to digital data transmission," IEEE Commun. Mag., vol. 13, no.2, pp. 15-20, Mar. 1975.

[OM] J.K. Omura, "Optimal receiver design for convolutional codes and channels with memory via control theoretical concepts," Info. Sci., vol. 3, pp. 243-266, July 1971.

[PR] J.G. Proakis, Digital Communication, 4th ed. New York:McGraw Hill, 2001.

[WJ] J.M. Wozencraft and I.M. Jacobs, Principles of Communication Engineering, Wiley New York, 1965.

[SK] B. Sklar, "How I Learned to Love the Trellis," IEEE Sig. Proc. Mag., May. 2003.

[FN2] G.D.Forney, Jr., "Maximum likelihood sequence estimation of digital sequences in the presence of intersymbol interference," IEEE Trans. Inform. Theory, vol. IT-18, PP. 363-378, May 1972.

7. Appendices

(1) The Viterbi Algorithm [HY], [FN1], [SK]

This section explains the steps in the Viterbi algorithm for Maximum Likelihood Sequence Estimation for an ISI channel.

As has been mentioned before, the objective of MLSD is to maximize the probability of the received sequence over all possible input sequences. It turns out that this boils down to finding the path through the trellis which minimizes the distance from the received sequence. We shall explore this statement further in this section.

If the ISI channel has a memory of L, it implies that the output depends on the current input bit as well as the L previous bits that were input. These L previous bits comprise the "state". Thus, there are 2^L possible states.

The first step is to create the trellis. The trellis for a channel memory of L=2 is shown in the figure. The transitions shown with a broken line are the transitions caused by an input bit 1 and the transitions shown with a solid line are the transitions caused by an input bit 0.

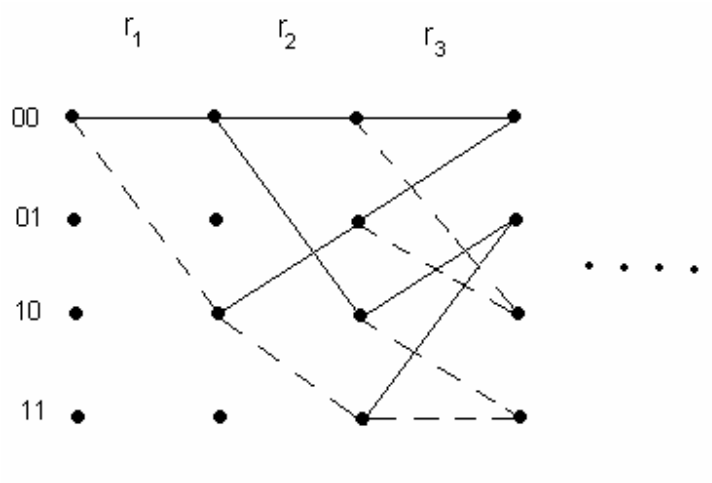

Fig 5. Trellis diagram

It is sometimes convenient to visualize state transitions as a shift register of size equal to the memory. As input bits come in from the left, the bits in the register shift to the right with the rightmost bit falling out. Thus if the state is currently 01 and the next input bit is a 1 then the transition will cause the new state to be 10.

Now each transition will also produce an output. This output is determined by the channel coefficients f_i. Note that the state represents the previous L bits. Thus, a transition will produce an output given by

<<<<<equation for rk = sigma(fi*s(k-i)) >>>>>

where the s(k-i) for i = 1, 2, ….L are simply the bits that define the current state of the trellis.

So as we traverse along any path in the trellis, each step we take produces some output which is calculated as we have shown above.

The Viterbi algorithm now simply uses an "Add-Compare-Select" method to choose the optimal path through the trellis and hence get an optimal estimate of the transmitted sequence. After the trellis stabilizes, each state at every stage has 2 paths coming into it. We have to eliminate one of these paths. The idea is that as we keep eliminating paths – keeping a single backward pointer at each state – at the end of the whole sequence we can simply trace backward along a single unique path since all sub-optimal paths have been eliminated. We make sure that we end up at the all-zero state by sending a string of zeros at the end of the message sequence. That way we know which final state we must start tracing backwards from.

The elimination of paths at a state is done by comparing the distance metrics for the two paths ending up at the same state. If a state k has two paths coming into it from states i and j in the previous stage then each of those transitions i-k and j-k have outputs X_i and X_j associated with them. Also, each of the states i and j in the previous stage have a distance metric associated with them – D_i and D_j – which is the distance of the paths up to there from the actual received sequence.

To eliminate paths at state k, we compare ($D_i + | X_i - r_i|$) and ($D_j + | X_j - r_j|$) and choose the lesser one. Thus, if j was the shosen state then while tracing backward we know that if we are at state k in this stage then the next step will be state j in the previous stage.

Once we have traversed the entire length of the trellis, stopping at every state to eliminate non-optimal paths we will end up at the all-zero state from which we now trace backwards along the optimal path, estimating a bit at each step of the way. In this fashion we estimate the path through the trellis which is a minimum distance away from the actual received sequence.

(2) Baseband equivalent discrete-time model for ISI channel using whitening filter

We use equivalent discrete-time model of intersymbol interference with WGN in this paper. The received signal which goes through this channel model consists of transmitted sequence and white gaussian noise and channel impulse response. But, if we use the optimum receiver which takes matched filter for convolution of channel impulse response with signaling waveform, the sampled value of the matched filter has a correlated gaussian noise samples.

For proving these noise sampled of matched filter to be correlated over adjacent samples, we can easily do that with the aid of this course's text book **[WJ]** and reference **[PK]**. For

easy analysis (we really need the property of white noise for MAP analysis), we have to use whitening filter. **[PK]**

In summary, the cascade of signaling waveform, the channel impulse response, the matched filter and the sampler, and after those processing, if we use the discrete-time noise whitening filter, all things could be represented as an equivalent discrete-time transversal filter having the set {fk} as its tap coefficients. The model looks like the following:

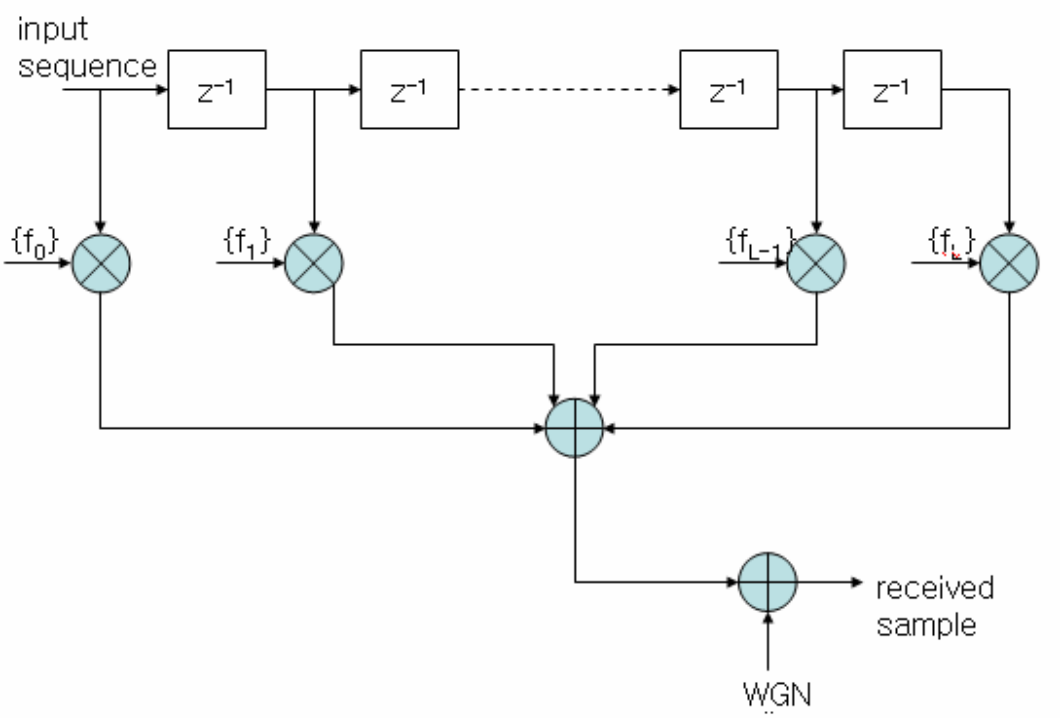

Figure 6. Equivalent discrete-time model of intersymbol interference channel with WGN.

Throughout the whole paper, we use this model for analysis and simulation. In simulation, we give the tap coefficients of the model (vector f) to the receiver in advance because we assume that the optimum MLSD receiver already knows about that. In analysis, we described these channel coefficients as g(i)

(3) Convolution Coding

The probability of error can be reduced by transmitting more bits than needed to represent the information being sent, and *convolving* each bit with neighbouring bits so that if one transmitted bit got corrupted, enough information is carried by the neighbouring bits to estimate what the corrupted bit was. This approach of transforming a number of *information* bits into a larger number of *transmitted* bits is called *channel coding*, and the particular approach of convolving the bits to distribute the information is referred to as *convolution coding*. The ratio of information bits to transmitted bits is the *code rate* (less than 1) and the number of information bits over which the convolution takes place is the *constraint length*.

For example, suppose you *channel encoded* a message using a *convolution code*. Suppose you transmitted 2 bits for every information bit (*code rate*=0.5) and used a *constraint length* of 3. Then the coder would send out 16 bits for every 8 bits of input, and each output pair would depend on the present and the past 2 input bits (*constraint length* =3). The output would come out at twice the input speed.

This is as seen in the below example:

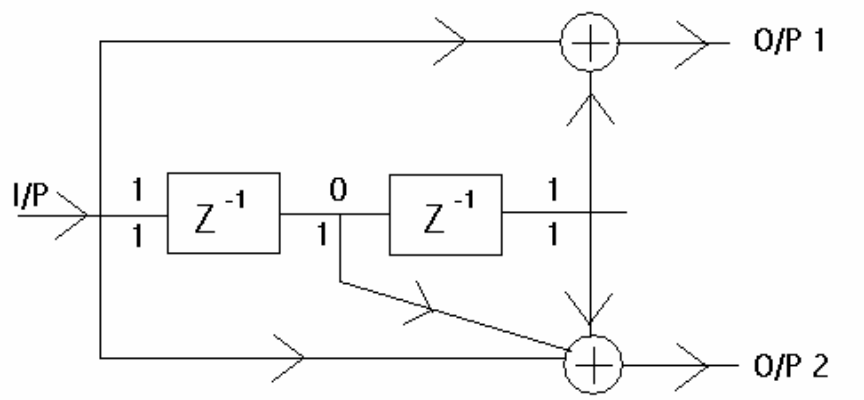

Fig. 7: The structure of encoder

In the above case, it is a convolution encoder of *code rate* 1/2 This means there are two output bits for each input bit.

The output z1 = x(n) ⊕x(n-1) ⊕x(n-2).

Here x(n) is the present input bit, x(n-1) was the previous (yesterdays) bit, etc.

The output z2= x(n) ⊕ x(n-2).

The input connections to the XORs can be written as binary vectors [1 1 1] and [1 0 1] are known as the *generating vectors* or *generating polynomials* for the code.

(4) MLSD for BPSK modulation for ISI with convolution coding

Main Program

```
%Revised: This has the integrated convolution encoding as well as the ISI
%channel together, with BPSK signaling

%Revised version: uses BPSK signaling with +1 and -1, keeping the signal
%power constant at 0dB. Thus the SNR is simply the negative of the noise
%power in dB.

%creates an ISI channel with known coefficients f(i) and hands the received
%output to a viterbi decoder for MLSD.

clear;
close all;

['Running....']

SNR=[];
biterr=[];
total_overall = [];
tot_ISI = [];

L = 2;                          %the memory
trel = poly2trellis(5,[23,35]);

pure_len = 500;                 %length f transmitted sequence without the leading and trailing
zeros
f = [ 0.4330  0.2165  ];      %coeff for most recent bit comes first

f_top = [ 0.2165    0.4330    0.8660 ] ; % for the Toplitz Matrix


%let's create the Toplitz Matrix with f_top

len = 2*pure_len + 2*L;

topl = zeros(len, len);
for i = 1:len - L

   temp = [ zeros(1,i-1) f_top zeros(1, len - i -L )];
   topl(i,:) = temp;

end
```

```
%let's create the Toplitz Matrix for the bit_by_bit sequence with f_top

x = pure_len + 2*L;

topl_plain = zeros(x,x);
for i = 1:x - L

      temp = [ zeros(1,i-1) f_top zeros(1, x - i -L )];
   topl_plain(i,:) = temp;

end

for N_dB=-1:-1:-10

total_over=0;
n=0;

while total_over<20
   n = n+1;
err_overall=0;

['-------------------------'];

%----------------do the convolutional encoding------------------

s_data = round( rand(1, pure_len-2) ); % 0s and 1s pure data bit sequence

s_data = [s_data 0 0]; %making sure the last 2 transmitted bits are zero

s_data_bpsk1 = sign( s_data - 0.5);

 % Define trellis.

s_encoded = convenc(s_data,trel); % Encode. This is twice the length of the pure data

s_enc = s_encoded;

%------------transmit as BPSK thru ISI + AWGN Channel the coded sequence and the
pure data--------------------------------

s_encoded = sign(s_encoded - 0.5); %converting 0 to -1 and 1 to +1 (BPSK)

filler = zeros(1,L) - 1; %a filler which is a string of -1's

s_encoded = [filler s_encoded filler];
```

```
%add leading and trailing zeros to make sure we start
%and end at the zeroth state

len = max(size(s_encoded));


r = topl*s_encoded'; %the received ISI coded sequence

r = r'; %making it a row vector

W = wgn(1, length(r), N_dB); %generating white gaussian noise samples with power
N_dB

r = r + W; %adding noise to the received samples

%-----------------Now pass it thru the Viterbi decoder for ISI--------------

[est_s_encoded, trellis1, distance1] = viterbi3_bpsk(r,L,f); %decodes the ISI for the
encoded sequence


%[est_s_plain, trellis12, distance2] = viterbi3_bpsk(r_pure,L,f); %decodes the ISI for the
plain data sequence

%est_s = [ zeros(1, L)-1 est_s];


x = length(est_s_encoded);
est_s_encoded(x-L) = -est_s_encoded(x-L);


for i=1:2*pure_len
   est_s2(i)=est_s_encoded(i);
end


% est_s2 now has the estimated encoded sequence after passing
%thru the ISI channel and its corresponding viterbi decoder

%---------convert this back to 0's and 1's

est_s2 = 0.5*(est_s2 + 1); %this is now the estimate of the convolutionally encoded
sequence in 0s and 1s

%-------------now pass this estimate of the encoded sequence thru the viterbi for
```

```
%convolutional decoding----------

tblen=18;


est_s_data=vitdec(est_s2,trel,tblen,'trunc','hard');

est_s_data(pure_len) = 0;  %making sure the last 2 bits match the transmitted sequence
est_s_data(pure_len-1) = 0;


err_overall = sum( abs(s_data - est_s_data) );

total_over=total_over+err_overall;


error=0;

trellis1;

end

total_overall = [total_overall total_over/(n*pure_len)]
SNR=[SNR -N_dB]
biterr = [biterr total_over/3*pure_len];
end

total_overall
```

Viterbi_bpsk.m :- Does the actual Viterbi decoding part.

```
%Revised version: uses BPSK signaling with +1 and -1, keeping the signal
%power constant at 0dB. Thus the SNR is simply the negative of the noise
%power in dB.

%This is going to be a viterbi decoder for ISI. Inputs are the received set of
%bits, the memory, the channel coefficients f(i) - a row matrix -  that determine how the
L
%previous bits affect the present bits. The output will (hopefully) be the
%maximum likelihood input sequence that caused this received sequence. The
%transmitted sequence must start from the all zero state and must end at
%the all zero state. This assumes 0's and 1's being transmitted even as
%voltages. Make changes if you want +1 and -1 for "inp_bit"

function [decoded, trellis, distance] = viterbi(r,L,f)
```

```
len = max(size(r)); %the number of received bits
st = 2^L; %number of states


f_0 = 0.8660; %the coefficient for the current bit

s = [0:st-1]; %a matrix with all the states numbered

trellis = zeros(st,len+1); %holds backward pointers as we go forward
distance = zeros(st, len+1); % keeps track of the distance metrics as we go along


% ----------------------------------------------------
%we now set the distance metrics for the points just before the stable
%states repeat. At this point, the state itself is the input sequence. The
%current bit can be obtained by looking at the MSB of the state. The

for k = 1:st

   %now we work backwards from each of these states to the all zero state
   %and sum up the distances

   state = k-1;

   for m = L:-1:1

      if state>=2^(L-1)
         inp_bit = 1;
      else
         inp_bit = -1;
      end

     ['Before shift'];

      state;

      state = bitshift(state,1,L); %to find out which state I came from

       ['After shift'];
      state;

      ['the output for the backward step is'];
      out = f*sign(nev_dec2bin(state, L) - 0.5)' + f_0*inp_bit ;
```

```
      distance(k,L+1) = distance(k,L+1) + abs(r(m) - out) ;

   end
end

distance;
% ---------------------------------------------------

%Now we proceed to calculate the distance metics stage by stage - for each of
%the stable states in each stage

for i = L+2:len+1,
   for j = 1:st

      x = s(j); %x now has the actual state number
      prev_1 = bitshift(x,1,L); %does a left shift. This is the prev state if the bit
                        %that fell out was a zero
      prev_2 = prev_1 + 1; %the prev state is the bit that fell out was a 1

      %we now figure out if the input bit to get to this state was a 1 or
      %a 0
      if x>=2^(L-1)
         inp_bit = 1;
      else
         inp_bit = -1;
      end

      out_1 = f*sign(nev_dec2bin(prev_1, L) - 0.5)' + f_0*inp_bit;

      % f should be such that the coefficient for the most recent bit is
      % first

      out_2 = f*sign(nev_dec2bin(prev_2, L) - 0.5)' + f_0*inp_bit;

       %now we have the 2 possible outputs for the two possible
       %transitions to this state

       d_1 = abs( r(i-1) - out_1) + distance(prev_1+1, i-1);
       d_2 = abs( r(i-1) - out_2) + distance(prev_2+1, i-1);

       %these are the 2 cumulative distance metrics for paths up to the
       %i'th stage

       if d_1 < d_2
          trellis(j,i) = prev_1; %pointing to the decided state in the previous stage
          distance(j,i) = d_1;
```

```
        else
            trellis(j,i) = prev_2;
            distance(j,i) = d_2;
        end


    end %for the j states
end %for the i stages


%now all we have to do is trace backwards from the last stage, which we
%made sure is the all zero stage by ending our transmitted sequence
%accordingly. The trace-back pointers are stored in 'trellis'. The trick
%to realise is that if I am in a state with a number >= 2^(L-1) then my
%step from that state to the previous state will imply a transmitted bit
%estimate of 1, otherwise it is an estimate of 0.

decoded = [];
temp_state = 0; %because we start tracing back from the 0th state
for p = len+1:-1:L+2

    if temp_state >= 2^(L-1)
        decoded = [1 decoded];
    else
        decoded = [-1 decoded];
    end

    temp_state = trellis(temp_state+1, p); %sets the state to the backward pointer in trellis

end

% we've traced backward to the first stable state. The rest of the sequence
% estimate is simply that state itself.

decoded = [sign(flip(nev_dec2bin(temp_state,L)) - 0.5) decoded];

% so decoded is now our estimate of the transmitted sequence of bits
```

Nev_dec2bin

```
%this function converts a decimal number into its binary equivalent in L bits and the
%result is a row matrix with each element as a bit
```

```
function [bin] = nev_dec2bin(x, L)

a = dec2bin(x,L);
bin = [];
for i = 1:L
   bin(i) = bin2dec(a(i));
end
```