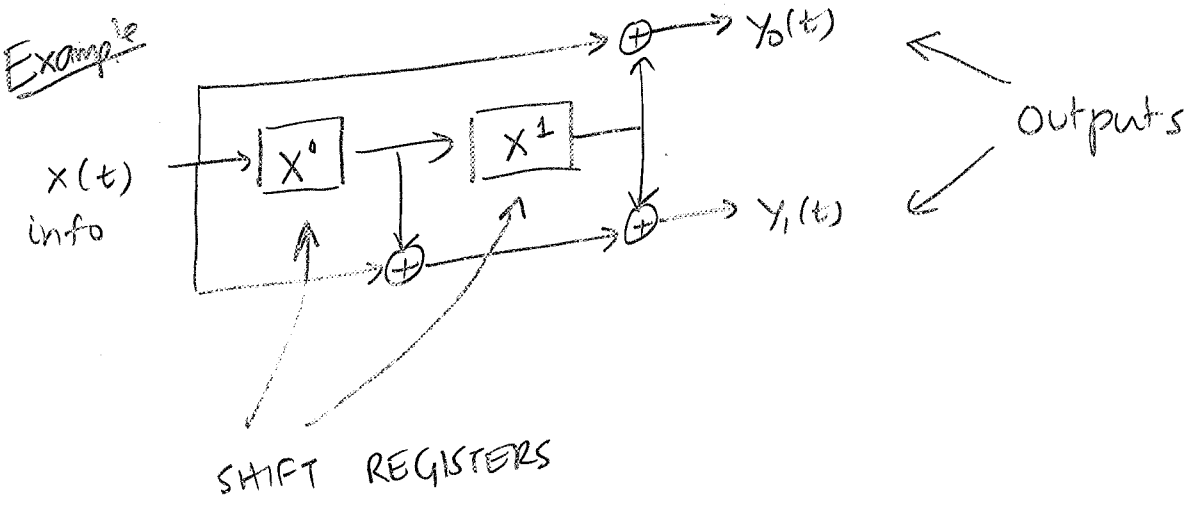
CHAPTER 4
CONVOLUTIONAL CODES

In Block codes, input was divided into blocks and then the blocks were coded.

Now we will code the whole input with one codeword, irrespective how long the input is.

These codes are called convolutional codes.

Example



$X(t) = 1011$

$Y_0(t) = 1001$

Input	X^0 X^1	Y_0	Y_1
1	0 0	1	1
0	1 0	0	1
1	0 1	0	0
1	1 0	1	0
0	1 1	1	
0	0 1	1	1
0	0	0	0

flushing not part of input

CODEWORD = (11, 01, 00, 10, 10, 11)

THIS is a convolutional code and its rate = $\frac{1}{2}$

Rate of convolutional code = $\frac{\#\text{ info bits/clock}}{\#\text{ output bits/clock}} = R_{cc}$

If we use our definition of rate of code for block codes, then we have 4 input bits and 12 output bits. So rate = $\frac{4}{12} = \frac{1}{3} < \frac{1}{2}$

This loss in rate is called fractional rate loss, and occurs because of flushing requirement.

In general,

L = # of info cycles

m = # flush cycles required

then block code rate

$$R_{BC} = \frac{L}{\frac{1}{R_{cc}}(L+m)} = R_{cc}\frac{L}{L+m} = R_{cc}\frac{1}{1+\frac{m}{L}}$$

If $L >> m$, then fractional rate loss $\frac{m}{L}$ is very small.

If L is small, then block codes are preferred.

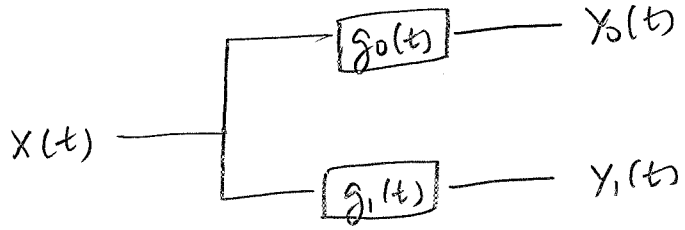
If L is large (compared to m), then convolutional codes are preferred.

Advantages of convolutional codes for $L \gg m$

(1) Flexibility w.r.t L.

(2) Convolutional codes have a nice soft-decision max-likelihood decoder.

EXAMPLE (CONTD)

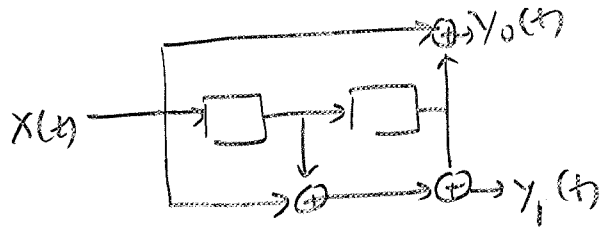


(For convolutional codes, we can use linear systems theory.)

1. This is a parallel system with each output linearly related to the input.

Let $\delta(t) = \{1, 0, 0, \ldots\}$ be the impulse function.

When you apply $x(t) = \delta(t)$, then each "arm" in the encoder can be represented by its "impulse response" $g_i(t)$.



$x(t) = 1\ 0\ 0\ 0\ 0\ \ldots$

$y_0(t) = g_0(t) = 1\ 0\ 1\ 0\ 0$

$y_1(t) = g_1(t) = 1\ 1\ 1\ 0\ 0$

For general inputs

CONVOLUTIONAL (MOD 2)

$$y_0(t) = x(t) \circledast g_0(t)$$

$$y_1(t) = x(t) \circledast g_1(t)$$

$$y_i(t) = x(t) \circledast g_i(t) = \sum_{\tau=0}^{\infty} x(\tau)\, g_i(t-\tau) = \sum_{\tau=0}^{\infty} g_i(\tau)\, x(t-\tau)$$

$$y_0(t) = \underset{1}{g_0(0)}\, x(t-0) + \underset{0}{g_0(1)}\, x(t-1) + \underset{1}{g_0(2)}\, x(t-2)$$

$$= x(t) + x(t-2)$$

$$y_1(t) = x(t) + x(t-1) + x(t-2)$$

] mod 2 arithmetic

EASIER TO SEE FROM THE ENCODER

So for rate $\frac{1}{n}$ code

$$y_i(t) = x(t) \circledast g_i(t)$$

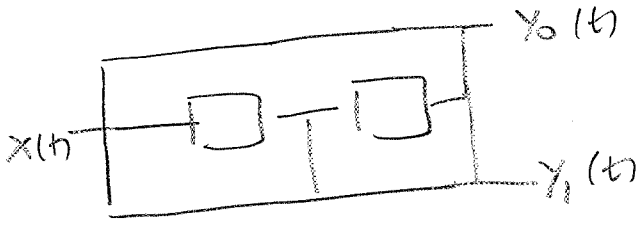
$g_i(t)$: i^{th} impulse response

$\circledast$: convolution (mod 2)

THIS IS TIME-DOMAIN REPRESENTATION OF ENCODING.

From Systems Theory, we know that we can also work in transform domain. Convolutions become multiplications.

EXAMPLE



$$X(t) = \{1,0,1,1\} \longleftrightarrow X(D) = 1 + D^2 + D^3$$

$$g_0(t) = \{1,0,1\} \longleftrightarrow G_0(D) = 1 + D^2$$

$$g_1(t) = \{1,1,1\} \longleftrightarrow G_1(D) = 1 + D + D^2$$

Then $Y_0(D) = X(D)\, G_0(D)$

$$= (1 + D^2 + D^3)(1 + D^2)$$

$$= 1 + D^2 + D^2 + D^4 + D^3 + D^5$$

$$= 1 + D^3 + D^4 + D^5 = 1001100\ldots$$

$$Y_1(D) = X(D)\, G_1(D)$$

$$= (1 + D^2 + D^3)(1 + D + D^2)$$

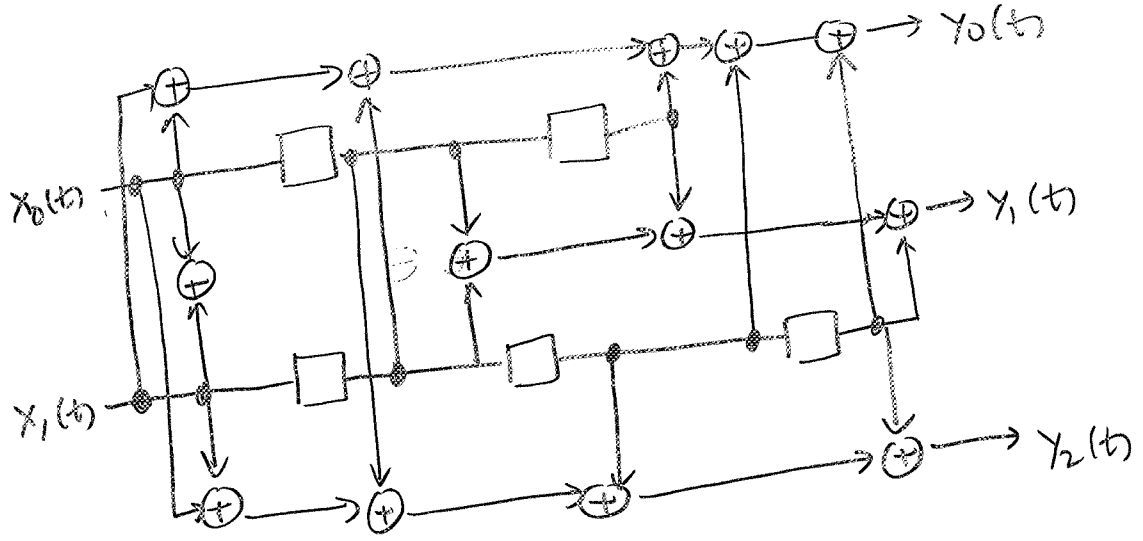
$$= 1 + D + D^2 + D^2 + D^3 + D^4 + D^3 + D^4 + D^5$$

$$= 1 + D + D^5 \longleftrightarrow 1\,1\,0\,0\,0\,1\,0\,0\ldots$$

output codeword $(11, 01, 00, 10, 10, 11, 00, 00, \ldots)$

FLUSHING IS AUTOMATIC

EXAMPLE A MORE ELABORATE EXAMPLE



$$\frac{\text{2 input bits/clock}}{\text{3 output bits/clock}} = \frac{2}{3} = R_{cc} \text{ (rate of conv. code)}$$

Again, we can define the impulse responses

Let $g_{ij}(t)$ = impulse response for i^{th} input to j^{th} output

Then $g_{00}(t) = y_0(t) \Big|_{\substack{x_0(t)=\delta(t) \\ x_1(t)=0}}$

$= \{1,0,1\} \longleftrightarrow G_{00}(D) = 1+D^2$

$g_{01}(t) = y_1(t) \Big|_{\substack{x_0(t)=\delta(t) \\ x_1(t)=0}}$

$= \{1,1,1\} \longleftrightarrow G_{01}(D) = 1+D+D^2$

$g_{02}(t) = \{1,1,0\} \longleftrightarrow G_{02}(D) = 1+D$

$g_{10}(t) = y_0(t) \Big|_{\substack{x_0(t)=0 \\ x_1(t)=\delta(t)}}$

$= \{1,1,1,1\} \longleftrightarrow G_{10}(D) = 1+D+D^2+D^3$

$g_{11}(t) = y_1(t) \Big|_{\substack{x_0(t)=0 \\ x_1(t)=\delta(t)}}$

$= \{1,1,0,1\} \longleftrightarrow G_{11}(D) = 1+D+D^3$

$g_{12}(t) = \{1,0,1,1\} \longleftrightarrow G_{12}(D) = 1+D^2+D^3$

ENCODING $\underline{X}(D) = (\underline{X}_0(D), \underline{X}_1(D))$

$\underline{Y}(D) = (\underline{Y}_0(D), \underline{Y}_1(D), \underline{Y}_2(D))$

We want

$$y_0(t) = x_0(t) \circledast g_{00}(t) + x_1(t) \circledast g_{10}(t)$$

$$y_1(t) = x_0(t) \circledast g_{01}(t) + x_1(t) \circledast g_{11}(t)$$

$$y_2(t) = x_0(t) \circledast g_{02}(t) + x_1(t) \circledast g_{12}(t)$$

$$\underline{Y}_0(D) = \underline{X}_0(D)\, G_{00}(D) + \underline{X}_1(D)\, G_{10}(D)$$

$$\underline{Y}_1(D) = \underline{X}_0(D)\, G_{01}(D) + \underline{X}_1(D)\, G_{11}(D)$$

$$\underline{Y}_2(D) = \underline{X}_0(D)\, G_{02}(D) + \underline{X}_1(D)\, G_{12}(D)$$

This is equivalent to the matrix equation

$$\underline{Y}(D)_{1\times 3} = \underline{X}(D)_{1\times 2}\, \mathbb{G}(D)_{2\times 3}$$

where $$\mathbb{G}(D) = \begin{bmatrix} G_{00}(D) & G_{01}(D) & G_{02}(D) \\ G_{10}(D) & G_{11}(D) & G_{12}(D) \end{bmatrix}$$

$$= [G_{ij}(D)]$$

ith input, jth output

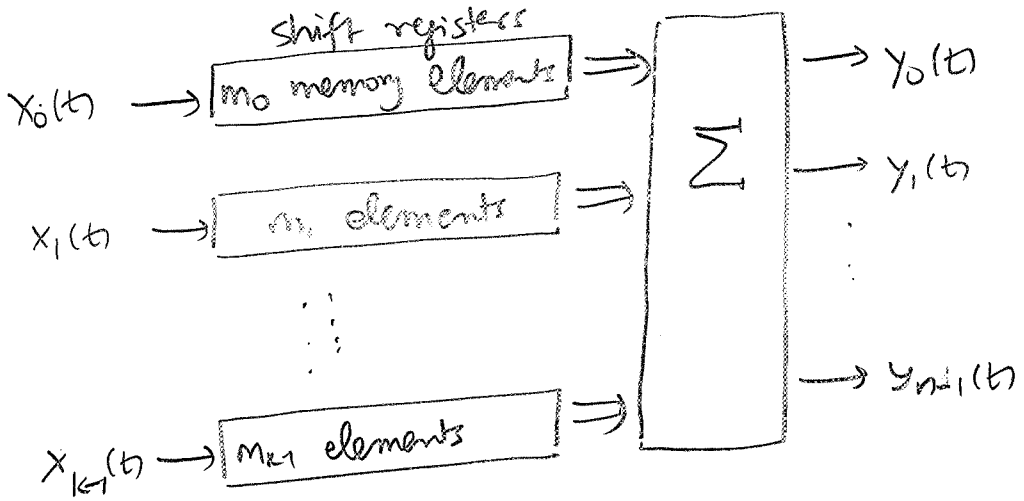
The matrix equation

$$\mathbb{Y}(D) = \mathbb{X}(D)\,\mathbb{G}(D)$$

is the generator matrix analog for convolution code.

GENERAL CASE

$$\text{Rate} = \frac{k}{n} = \frac{\#\text{ of input bits / clock}}{\#\text{ of output bits / clock}}$$



Then $$\underset{1\times n}{\mathbb{Y}(D)} = \underset{1\times k}{\mathbb{X}(D)}\;\underset{k\times n}{\mathbb{G}(D)}$$

DEFINE $$M = \sum_{i=0}^{k-1} m_i = \text{total \# of memory elements}$$

determines the # of distinct states of the encoder (2^M different states)

DEFINE $m = \max_{0 \le i \le k-1} \{m_i\}$ = maximal memory order

Determines the length of flushing

DEFINE $K = 1+m$ is the Constraint length.

(BEWARE: NO STANDARD DEFINITION)

NEXT TIME: FINITE MACHINE REPRESENTATION

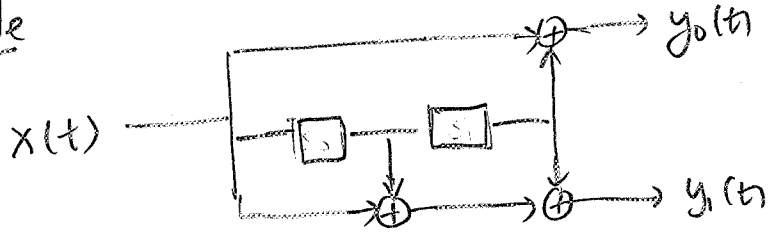
CONVOLUTIONAL CODES

LAST time we looked at general convolutional encoding in time and transform domain. In a general rate $\frac{K}{n}$ convolutional code, there are $M = \sum_{i=0}^{K-1} m_i$ shift registers

$\Rightarrow$ there are 2^M possible states

And thus a natural representation of convolutional codes is as finite state machines.

Example



The state of the shift register is the 2-tuple of contents of the memory elements

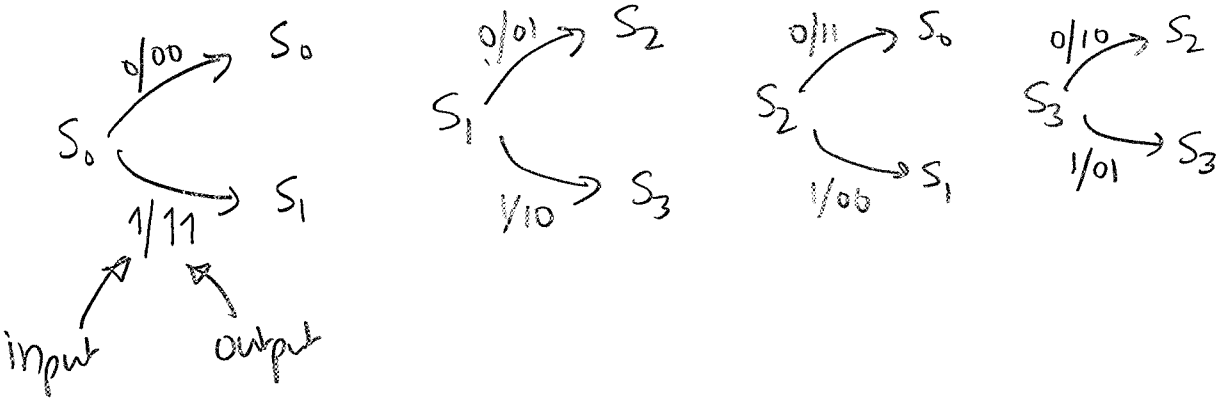
$S_0 = (0,0)$

$S_1 = (1,0)$

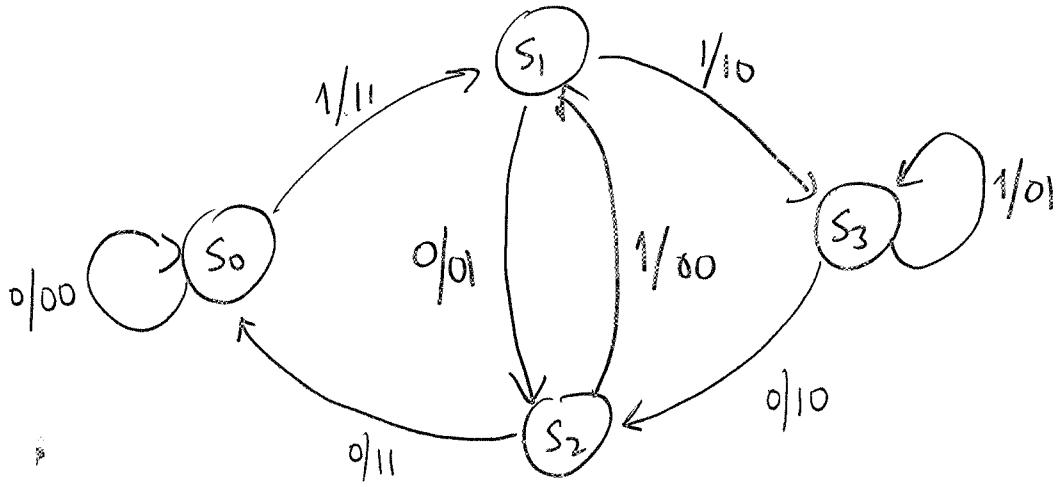
$S_2 = (0,1)$

$S_3 = (1,1)$

If the shift register is in state S_0, then it transitions to S_0 if the input is 0 else to S_1.



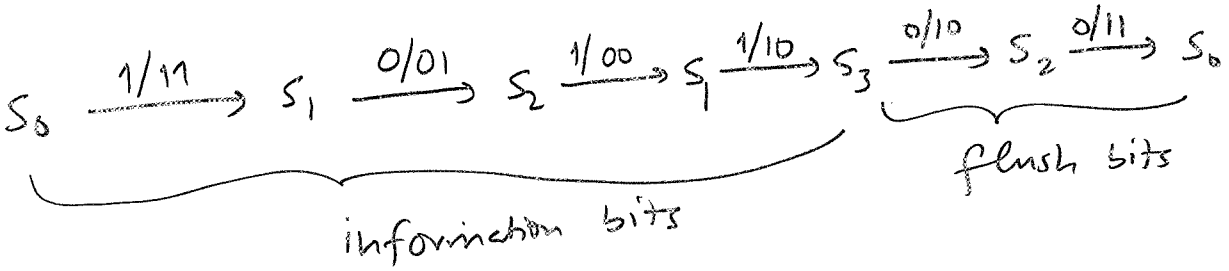
You can combine the above into a compact state diagram.



This is an alternate representation of encoder.

Consider input $x(t) = 1011$

RECALL : encoder always starts from state S_0 and is returned to state S_0 by including m flush bits (all 0's).



$\bar{c} = (11, 01, 00, 10, 10, 11)$

Third representation of encoding

(1) time domain using convolution

(2) transform domain using poly multiplication

MAXIMUM LIKELIHOOD DECODING

Instead of Binary Symmetric Channel (BSC) we will be interested in Additive White Gaussian Noise (AWGN) Channel.

For AWGN, we will need modulation.

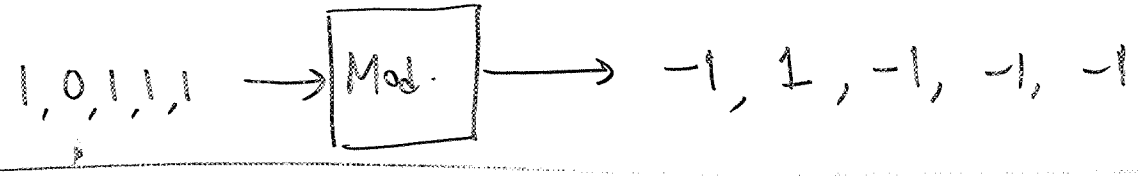
Use the simplest case

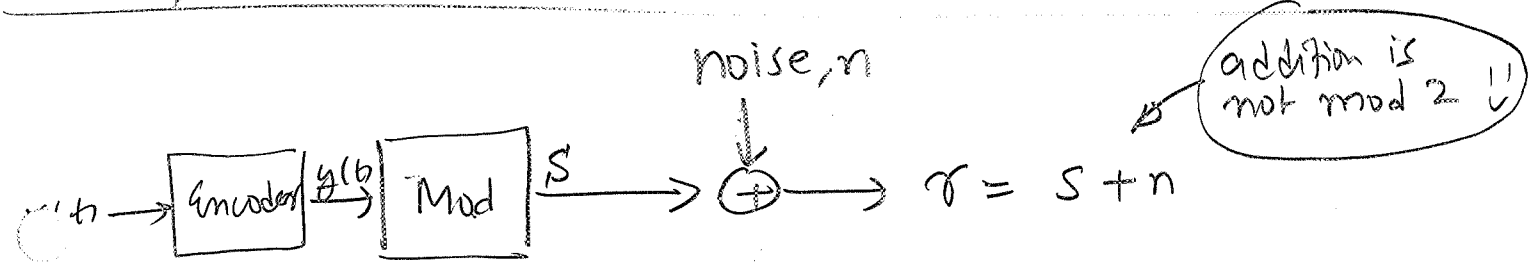
BPSK : Binary Phase Shift Keying

$X = 0, 1$ input symbol to modulator

$S = (-1)^X$ output of modulator

$$= \begin{cases} +1 & X = 0 \\ -1 & X = 1 \end{cases}$$





noise n is
- statistically independent for each modulated symbol
- identically distributed as Gaussian with mean 0 & variance σ^2.

So

Input to Encoder : $\{\bar{x}_t\}_{t=0}^{L-1}$ ← L clocks of shift register, $\bar{x}_t$ is a k-tuple

Output of Encoder ($\frac{k}{n}$ convolutional code) : $\{\bar{y}_t\}_{t=0}^{L+m-1}$ ← each $\bar{y}_t$ is an n-tuple

Receive : $\{\bar{r}_t\}_{t=0}^{L+m-1}$ ← $\bar{r}_t$ is an n-tuple

$\bar{r}_t = \bar{s}_t + \bar{n}_t$ ← AWGN noise

$\bar{s}_t = (-1)^{\bar{y}_t}$

GOAL : To find the input sequence $\{\hat{\bar{x}}_t\}_{t=0}^{L-1}$ which maximizes

$$\text{Prob}\left[\{\bar{x}_t\}_{t=0}^{L-1} \text{ is sent} \mid \{\bar{r}_t\}_{t=0}^{L+m-1}\right]$$

Maximum Aposteriori (MAP) DECODER

THIS is the aposteriori probability of $\{\bar{x}_t\}_{t=0}^{L-1}$ having received $\{\bar{r}_t\}_{t=0}^{L+m-1}$. Generally tends to be complex since PDF is not easy to to derive.

If the input information sequences $\{\bar{x}_t\}_{t=0}^{L-1}$ are equally likely, then MAP decoder is equivalent to maximizing likelihood (ML)

$$\text{Prob}\left[\{\bar{r}_t\}_{t=0}^{L+m-1} \mid \{\bar{x}_t\}_{t=0}^{L-1}\right]$$

Because according to BAYES RULE

$$\text{Prob}\left[\{\bar{r}_t\} \mid \{\bar{x}_t\}\right] \text{Prob}\left[\{\bar{x}_t\}\right] \leftarrow \text{constant, so independent of } \{\bar{x}_t\}$$

$$= \text{Prob}\left[\{\bar{x}_t\} \mid \{\bar{r}_t\}\right] \text{Prob}\left[\{\bar{r}_t\}\right]$$

↑ independent of $\{\bar{x}_t\}$

Thus

$$\text{Prob}\left[\{\bar{r}_t\} \mid \{\bar{x}_t\}\right] = \alpha \, \text{Prob}\left[\{\bar{x}_t\} \mid \{\bar{r}_t\}\right]$$

↑ true constant independent of $\{\bar{x}_t\}$

So maximizing RHS ≡ maximizing LHS

ML decoder is much easier since the probability distribution involved is simply Gaussian distribution

NOTE

$$\text{Prob}\left(\{\bar{r}_t\}_{t=0}^{L+m-1} \,\middle|\, \{\bar{x}_t\}_{t=0}^{L-1}\right) = \text{Prob}\left(\{\bar{r}_t\}_{t=0}^{L+m-1} \,\middle|\, \{\bar{s}_t\}_{t=0}^{L+m-1}\right)$$

↑ 1-1 relation between info bits and transmitted symbols

$$= \prod_{t=0}^{L+m-1} \text{Prob}\left(\bar{r}_t \,\middle|\, \{\bar{s}_t\}_{t=0}^{L+m-1}\right)$$

↑ independent noise samples

$$= \prod_{t=0}^{L+m-1} \text{Prob}\left(\bar{r}_t \,\middle|\, \bar{s}_t\right)$$

↑ $\bar{r}_t$ only depends on $\bar{s}_t$

GAUSSIAN!

The primary calculation

$$p\left(\{\bar{r}_t\} \,\middle|\, \{\bar{s}_t\}\right) = \prod_{t=0}^{L+m-1} p\left(\bar{r}_t \,\middle|\, \bar{s}_t\right)$$

↑ this is n-tuple

We can break it even further

$$p\left(\bar{r}_t \,\middle|\, \bar{s}_t\right) = \prod_{i=0}^{n-1} p\left(\bar{r}_t^{(i)} \,\middle|\, \bar{s}_t^{(i)}\right)$$

$\bar{s}_t^{(i)}$ ← i^{th} coordinate of vector $\bar{s}_t$

↑ all noise samples are iid for each BPSK use.

So ML decode does the following

(1) Enumerate all info sequences $\{\bar{x}_t\}_{t=0}^{L-1}$ and corresponding modulated coded sequences $\{\bar{s}_t\}_{t=0}^{L+m-1}$

(2) For each, compute $P\left(\{\bar{r}_t\}_{t=0}^{L+m-1} \mid \{\bar{s}_t\}_{t=0}^{L+m-1}\right)$

(3) Store the sequence with maximum value of above likelihood and call it $\{\hat{\bar{s}}_t\}_{t=0}^{L+m-1}$

(4) Output $\{\hat{\bar{x}}_t\}_{t=0}^{L-1}$ corresponding to $\{\hat{\bar{s}}_t\}_{t=0}^{L+m-1}$

So this is a brute force search and useful only for small L. But for convolutional codes to be useful, $L \gg m > 1$. !!

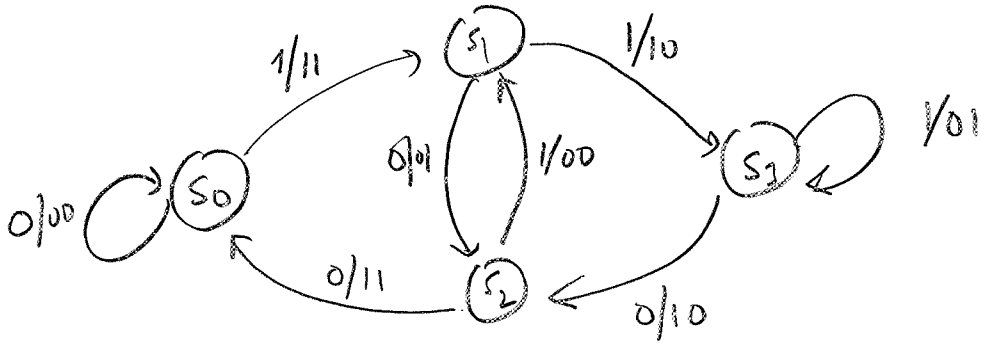
DON'T DESPAIR $P(\{\bar{r}_t\} | \{\bar{s}_t\})$ depends only on $P(\bar{r}_t | \bar{s}_t)$, so a clever bookkeeping is possible.

BUT we will need a (yet another) different representation of convolutional encoding.

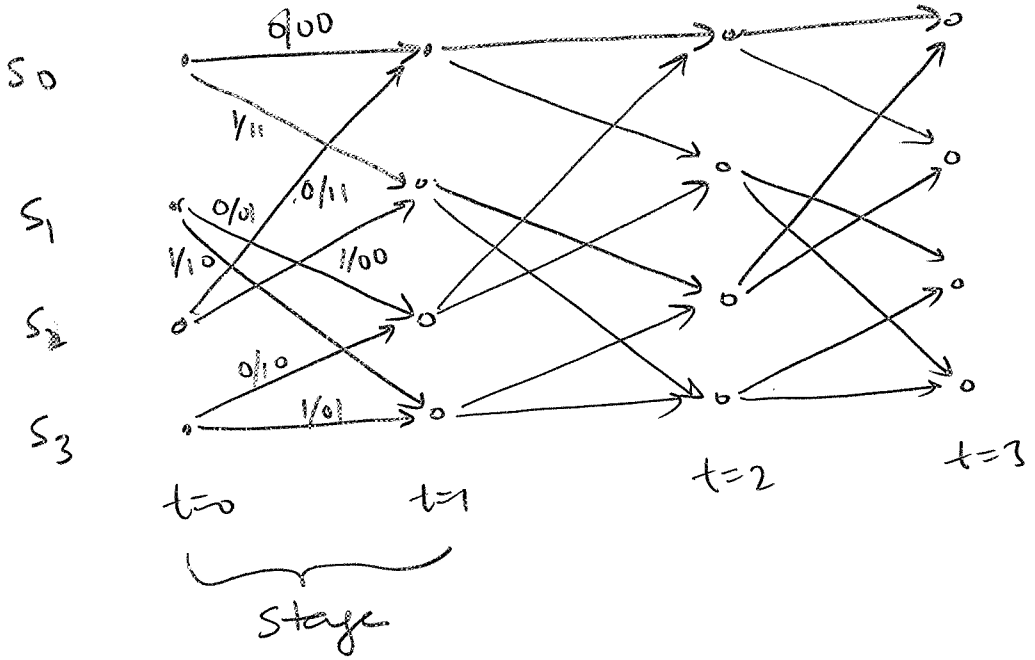
The state diagram gives a static representation, What we need is a dynamic representation, showing the evolution of the encoding function.

A time dependent representation of the state diagram is "the trellis".

Example



Each clocking of the encoder is shown by a separate state diagram

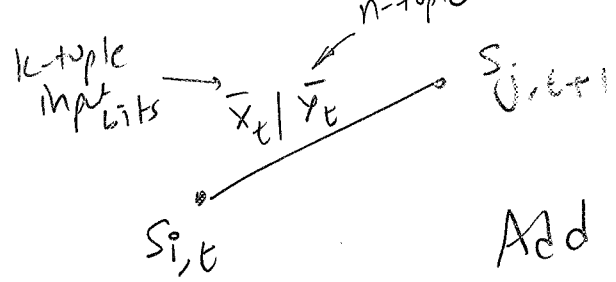


Q: Within the trellis representation, what do codewords correspond to?

A: They correspond to unique paths beginning in S_0 and ending in S_0 after L+m stages.

Every codeword has a unique such path and every path in the trellis has a unique corresponding codeword.

Let's use the trellis to implement a ML decoder

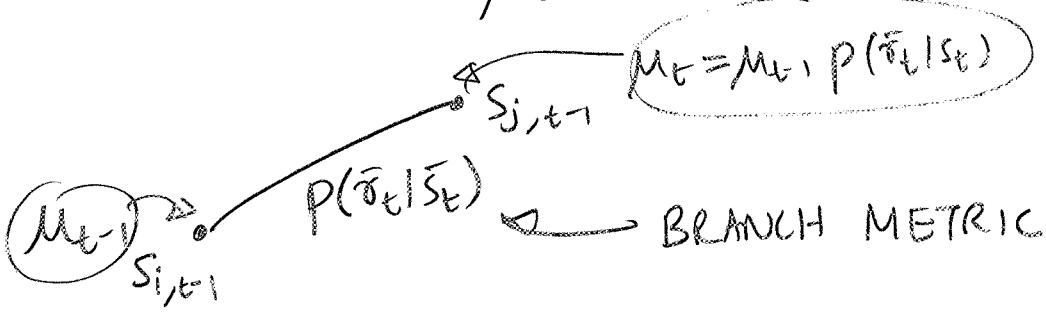


Add another label $p(\bar{r}_t | \bar{s}_t)$

So: Given any codeword path through the trellis, I store at each node of the path the partial likelihood function

$$\mu_t = \prod_{\tau=0}^{t} p(\bar{r}_\tau | \bar{s}_\tau)$$

$$= \mu_{t-1} p(\bar{r}_t | \bar{s}_t)$$



This gives the following decoding algorithm

(1) Enumerate all paths through the trellis that correspond to codewords

(2) For each, compute partial likelihoods recursively from stage to stage

(3) Select path with largest full likelihood.

Looks fancier,

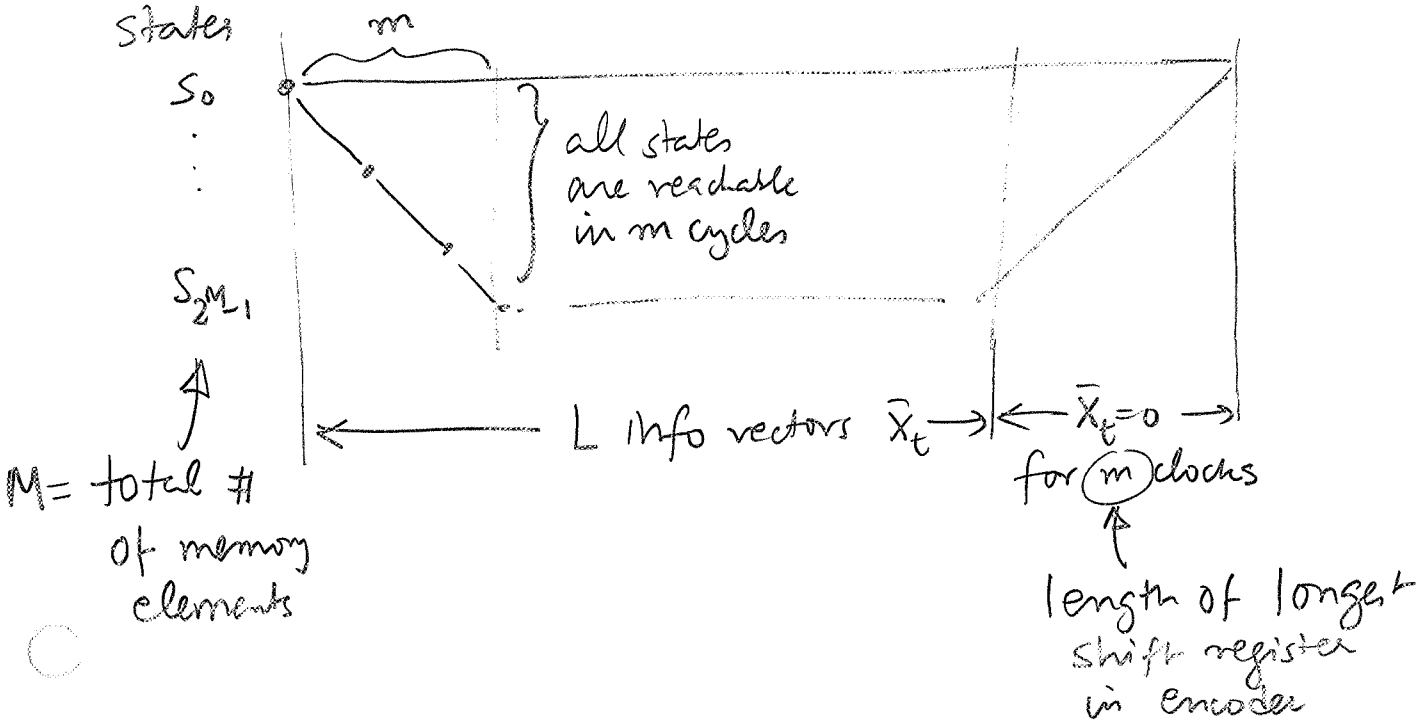
Feels fancier,

But still brute force!

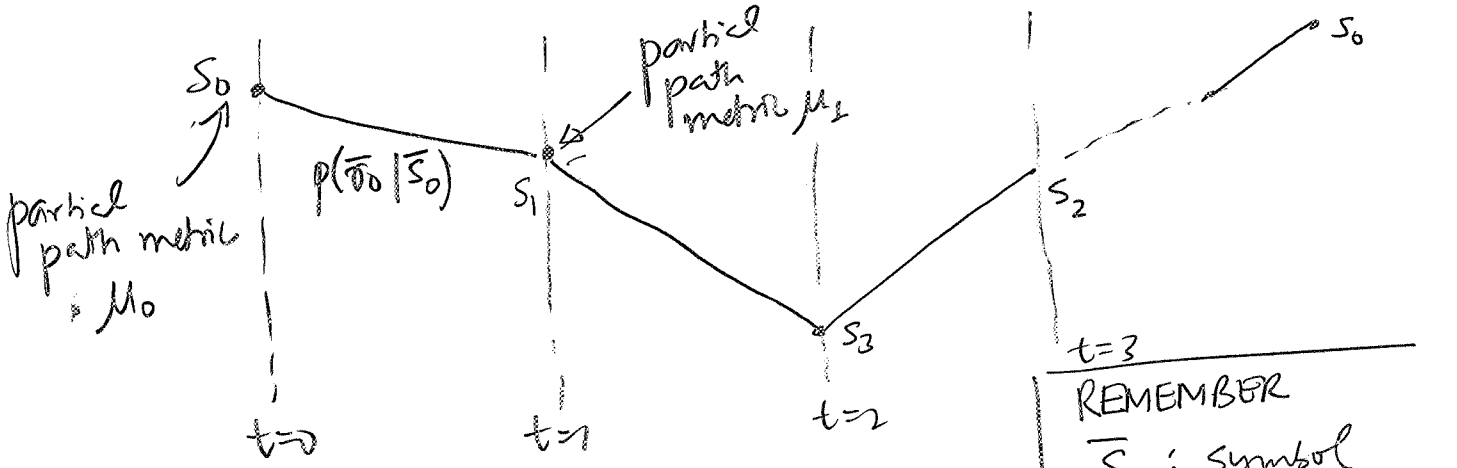
This is a ML decoder with GUI.

NEXT TIME – THE TRICK!

Last time, we saw that encoding can be represented as a trellis.



Each branch is labeled with the path metric



REMEMBER
$\bar{s}_t$: symbol
S_i : state
NOT SAME!

Then $\mu_1 = \mu_0 \underbrace{p(\bar{r}_0|\bar{s}_0)}_{\text{branch metric}}$

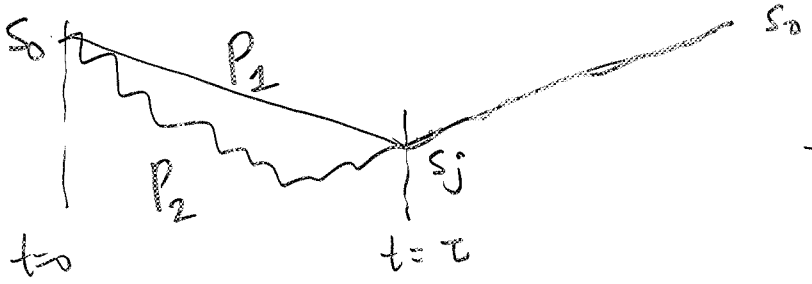
updated partial path metric

old partial path metric

In general $\mu_t = \mu_{t-1} P(\bar{r}_{t-1} \mid \bar{s}_{t-1})$ ... (*)

We initialize $\mu_0 = 1$ and proceed to update along the path using (*).

THE TRICK : Consider what happens when two paths merge in the trellis.



Two different paths upto $t=\tau$. After that, they are same.

Suppose $\mu_\tau(P_1) > \mu_\tau(P_2)$

i.e. path metric for path P_1 is more than path metric for path P_2 upto $t=\tau$

Claim: The ML path does not include P_2 (upto $t=\tau$)

Proof: Suppose ML path includes $P_2(\tau)$. Then ML path include a tail P_{tail} from s_j to s_0 such that $(P_2(\tau), P_{tail})$ has the highest likelihood.

But note that $(P_1(\tau), P_{tail})$ is also a valid codeword and has higher likelihood. So ML path cannot include $P_2(\tau)$.

BOTTOM LINE: We don't have to consider all paths. As you proceed through the trellis, you can discard some of the initial segments.

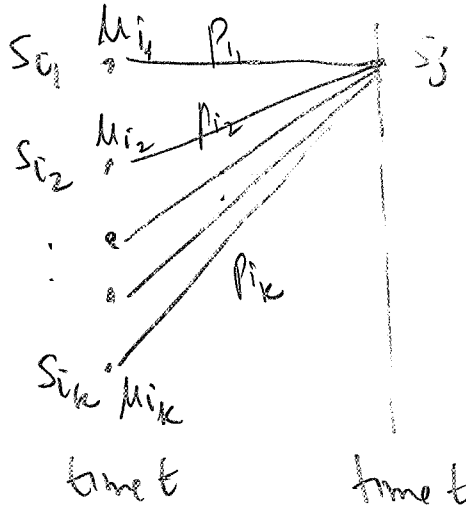
THIS TRICK WAS IDENTIFIED BY VITERBI & HENCE THIS DECODING IS LABELED VITERBI DECODER

VITERBI DECODING

(1) Assign partial path metric $\mu_0 = 1$ at node S_0 at time $t=0$.

(2) At time t, compute branch metric for all transitions between states at time t and states at time $t+1$.

(3) Update partial path metrics for all states at time $t+1$. (Note: for a given node, there are 2^K incoming branches)



(4) Store in S_j $\max\{\mu_i P_{ie}\}$, and deactivate all incoming paths except the one with largest $\mu_i P_{ie}$. Thus at time $t+1$, each node is associated with only one "survivor" path. (You do it for all nodes and keep only the surviving path).

(5) If the end of the trellis is reached, there is only one node, S_0, and hence only one surviving path (ML path).

If end of trellis is not reached, increment t and go to step (2).

(6) Traceback the ML path back through the trellis and output the info vectors $\bar{X}_0$ corresponding to each branch in the ML path.

So ML decoding of convolutional codes can be accomplished with Viterbi decoding and branch metrics

$$p(\bar{r}_0 | \bar{s}_t)$$

Now $$p(\bar{r}_t|\bar{s}_t) = \prod_{i=0}^{n-1} p(r_t^{(i)}|s_t^{(i)})$$

where $\bar{r}_t = (r_t^{(0)}, r_t^{(1)}, \ldots, r_t^{(n-1)})$ is the received n-tuple of noisy channel symbols.

For i.i.d Gaussian noise $N(0, \sigma^2)$ and BPSK modulation

$$p(\bar{r}_t|\bar{s}_t) = \prod_{i=0}^{n-1} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left\{-\frac{1}{2\pi\sigma^2}(r_t^{(i)} - s_t^{(i)})^2\right\}$$

$$= \left(\frac{1}{\sqrt{2\pi\sigma^2}}\right)^n \exp\left\{-\frac{1}{2\sigma^2}\sum_{i=0}^{n-1}(r_t^{(i)} - s_t^{(i)})^2\right\}$$

$$= \frac{1}{(2\pi\sigma^2)^{n/2}} \exp\left\{-\frac{1}{2\sigma^2}\|\bar{r}_t - \bar{s}_t\|_2^2\right\}$$

squared Euclidean distance between real valued vectors $\bar{r}_t$ & $\bar{s}_t$.

Similarly the path metrics are

$$p\left(\{\bar{r}_t\}_{t=0}^{L+m-1} \,\middle|\, \{\bar{s}_t\}_{t=0}^{L+m-1}\right) = \prod_{t=0}^{L+m-1} p(\bar{r}_t|\bar{s}_t)$$

$$= \frac{1}{(2\pi\sigma^2)^{n(L+m)}} \exp\left\{\frac{-1}{2\sigma^2} \sum_{t=0}^{L+m-1}{}' \|\bar{r}_t - \bar{s}_t\|^2\right\}$$

$$= \frac{1}{(2\pi\sigma^2)^{n(L+m}} \exp\left\{\frac{-1}{2\sigma^2} \left\| \{\bar{r}_t\}_{t=0}^{L+m-1} - \{\bar{s}_t\}_{t=0}^{L+m-1} \right\|^2\right\}$$

Euclidean distance

Take the log of both sides

$$\log(\text{path metric}) = \log\left\{\frac{1}{(2\pi\sigma^2)^{n(L+m}}\right\} - \frac{1}{2\sigma^2} \left\| \{\bar{r}_t\}_{t=0}^{L+m-1} - \{\bar{s}_t\}_{t=0}^{L+m-1} \right\|^2$$

NOTE: If Path A has higher path metric than Path B, then Path A has higher log(path metric) than Path B. (log is monotonically increasing function)

Also

$$\log(\text{path metric}) = \log \prod_{t=0}^{L+m-1} p(\bar{r}_t|\bar{s}_t) = \sum_{t=0}^{L+m-1}{}' \log p(\bar{r}_t|\bar{s}_t)$$

↑ log of branch metric

So in our Viterbi algorithm, we could have used log branch metrics — accumulated by adding.

NOTE $$p(\bar{r}_t|\bar{s}_t) = \underbrace{\log \frac{1}{(2\pi\sigma^2)^n}}_{\text{same for all input branches}} - \frac{1}{2\sigma^2}\|\bar{r}_t - \bar{s}_t\|^2$$

So we can use $-\|\bar{r}_t - \bar{s}_t\|^2$ as branch metrics

$$\text{Max } \log p(\bar{r}_t|\bar{s}_t) \Leftrightarrow \text{Max } -\|\bar{r}_t - \bar{s}_t\|^2 \Leftrightarrow \text{Min } \|\bar{r}_t - \bar{s}_t\|^2$$

For Gaussian channels (i.i.d), ML is same as finding the modulated codeword which is closest to the received signal in Euclidean distance.

PERFORMANCE ANALYSIS

CONVOLUTIONAL CODES ARE LINEAR

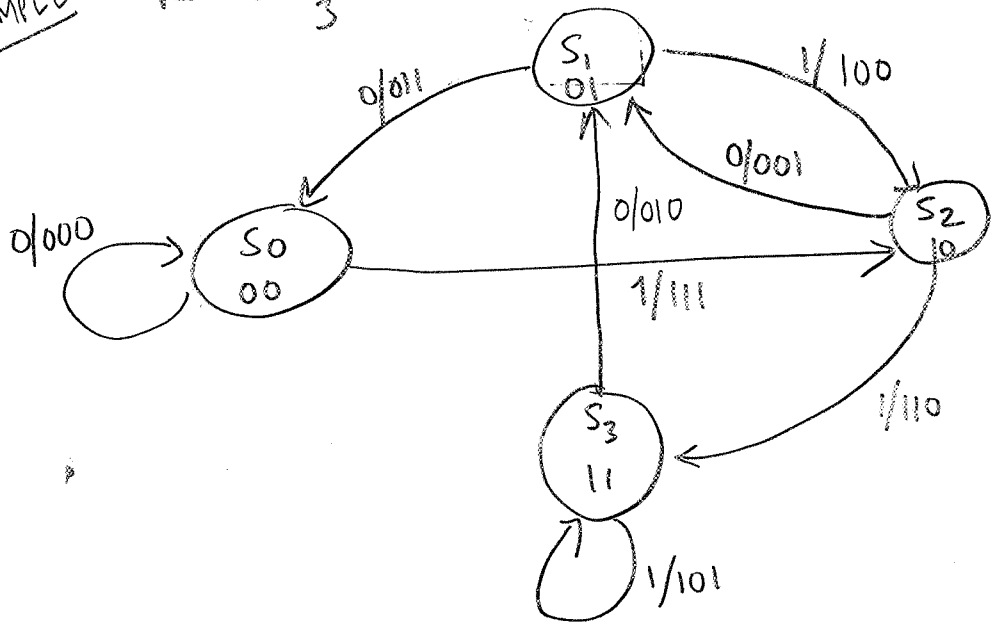
$\Rightarrow$ set of Hamming distances between codewords

= set of Hamming distance from all-zero codeword

= set of weights of different codewords

The state diagram of convolutional encoder can be used to get the ~~full~~ weight spectrum of the convolutional code.

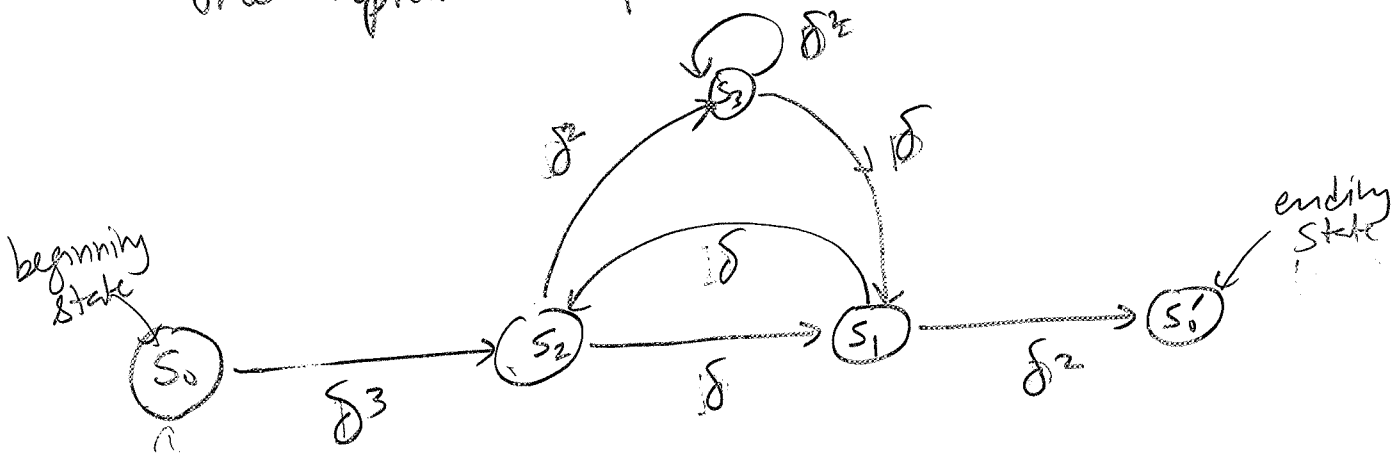
EXAMPLE Rate $\frac{1}{3}$ code with $m = 2$



(1) Relabel all branches of the state diagram as either $\delta^0=1, \delta^1, \delta^2, \delta^3$ where the exponent of δ = Hamming weight of the output (= Hamming distance from all zero codeword)

(2) Remove self loop at node S_0, since it does not contribute to distance spectrum.

(3) Split node S_0 into two nodes, one represents input and other output.



(4) Write the state equations

$$\left.\begin{aligned} X_2 &= \delta^3 X_0 + \delta X_1 \\ X_1 &= \delta X_3 + \delta X_2 \\ X_3 &= \delta^2 X_2 + \delta^2 X_3 \\ X_0' &= \delta^2 X_1 \end{aligned}\right\} (\dagger)$$

The Transfer Function (Generating Function) of the code is defined as

$$T(\delta) = \frac{X_0'}{X_0}$$

Solving (†), we get

$$T(\delta) = \frac{\delta^6}{1-2\delta^2}$$

$$= \delta^6 + 2\delta^8 + 4\delta^{10} + 8\delta^{12} + \cdots$$

$$= \sum_{d=6}^{\infty} a_d \delta^d$$

where $a_d = \begin{cases} 2^{(d-6)/2} & d \text{ is even} \\ 0 & d \text{ is odd} \end{cases}$

$T(\delta)$ indicates

(1) one single path with weight = 6 (or distance = 6 from all zero codeword)

(2) two paths with weights = 8

(3) 4 paths with weight = 10 ....

The minimum distance of code is called the minimum free distance d_{free}.

Here $d_{free} = 6$

d_{free} represents how long the path can be away from all zero path before (free from it) returning back to it.

Higher d_{free} generally means better error probability performance (d_{free} is like d_{min}).

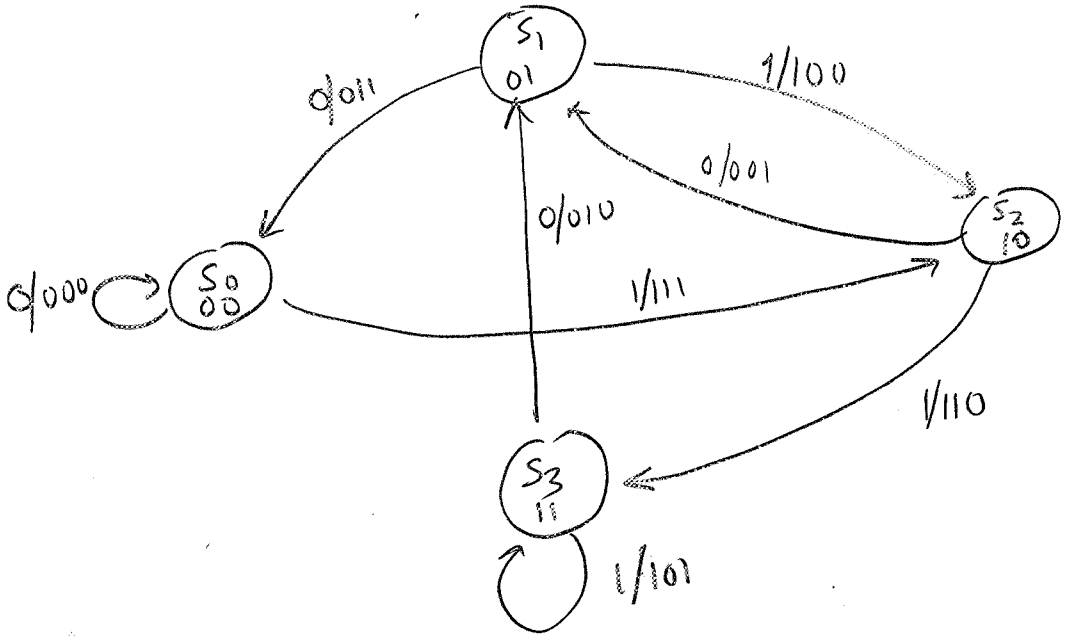
Next time - Probability of error bounds

PERFORMANCE ANALYSIS

REVIEW FROM LAST TIME

We saw that state diagram can be used to obtain the weight spectrum of convolutional codes.
Let's look at the same example

EXAMPLE Rate $\frac{1}{3}$ code with $m=2$

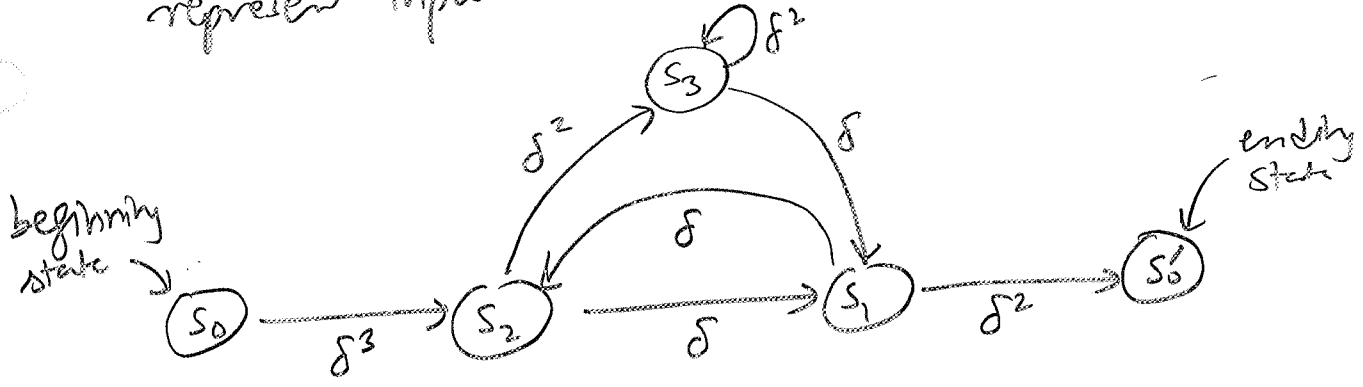


(1) Relabel all branches of the state diagram as either $\delta^0 = 1, \delta^1, \delta^2, \delta^3$ where the

exponent of δ = Hamming weight of the output
(= Hamming distance from all zero codeword)

(2) Remove self loop at s_0. (since it does not contribute to distance spectrum)

(3) Split node s_0 into two nodes, one will represent input and the other output.



(4) Write the state equations (how you arrive at a state)

$$X_2 = \delta^3 X_0 + \delta X_1$$
$$X_1 = \delta X_3 + \delta X_2$$
$$X_3 = \delta^2 X_2 + \delta^2 X_3$$
$$X_0' = \delta^2 X_1$$

} (*)

The Tranfer function (or Generating function or Path Enumerator) of the code is defined as

$$T(\delta) = \frac{X_0'}{X_0} \quad \left(\frac{\text{Output}}{\text{Input}} \right)$$

Solving the set of equations (*), we get

$$T(\delta) = \frac{\delta^6}{1-2\delta^2}$$

$$= \delta^6 + 2\delta^8 + 4\delta^{10} + 8\delta^{12} + \cdots$$

$$= \sum_{d=6}^{\infty} a_d \delta^d$$

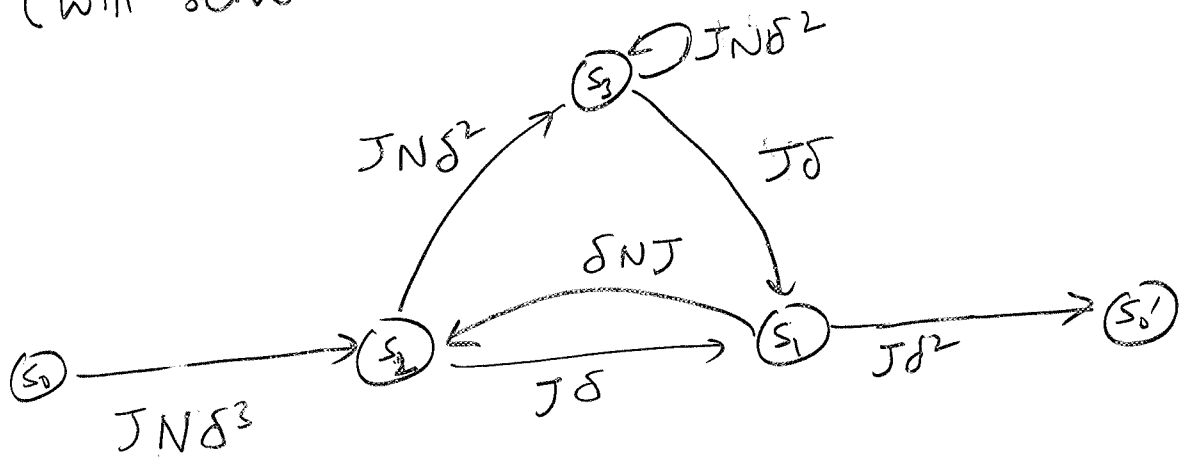
where $a_d = \begin{cases} 2^{(d-6)/2} & d \text{ is even} \\ 0 & d \text{ is odd} \end{cases}$

- Exponent of smallest power is called Minimum free Distance (d_{free})
  (Here $d_{free} = 6$)
- Enumerates all paths which leave and merge into S_0 only once.
- Can be used to find actual weight distribution for a given block length.
- We will use to derive a useful bound on performance

This approach is even more useful than you think!

* Suppose we add another label N for all branch transitions caused by 1.

* Add another factor J to each exponent (will serve to count the # of branches in a path)



The new state equations are

$$\left.\begin{aligned} X_2 &= JN\delta^3 X_0 + JN\delta X_1 \\ X_1 &= J\delta X_2 + J\delta X_3 \\ X_3 &= JN\delta^2 X_3 + JN\delta^2 X_2 \\ X_0' &= J\delta^2 X_1 \end{aligned}\right] \quad (*)$$

Solving (*) gives

$$T(\delta, N, J) = \frac{J^3 N \delta^6}{1 - JN\delta^2(1+J)}$$

$$= J^3 N \delta^6 + J^4 N \delta^8 + J^5 N^2 \delta^8 + J^5 N^3 \delta^{10} + 2J^6 N^3 \delta^{10} + J^7 N^3 \delta^{10} + \cdots$$

- weight = 6 (exponent of δ)
- path length = 3 (exponent of J)
  - ⇒ 3 information bits
- only one info bit was 1

For $J = N = 1$, we get our original transfer function $T(\delta)$.

Performance bounds for Convolutional Codes (BSC channel)

Consider the code shown on the next page.

$$T(\delta, N, J) = \frac{\delta^7 N J^3}{1 - \delta N J (1 + \delta^2 J)}$$

$$= \delta^7 N J^3 \left[1 + \delta N J (1 + \delta^2 J) + \delta^2 N^2 J^2 (1 + \delta^2 J^2) + \cdots\right]$$

$$= \delta^7 N J^3 + \delta^8 N^2 J^4 + \delta^9 N^3 J^5 + \delta^{10} (N^2 J^5 + N^4 J^5) + \cdots$$

So there is one codeword of weight 7 and length 3 generated by an information stream of weight = 1. (shown in dark on next page)

DEFN: A FIRST ERROR EVENT is made at an arbitrary time j if the all-zero path (the correct path) is eliminated for the first time at time j for another path (the incorrect path) entering the all-zero state.

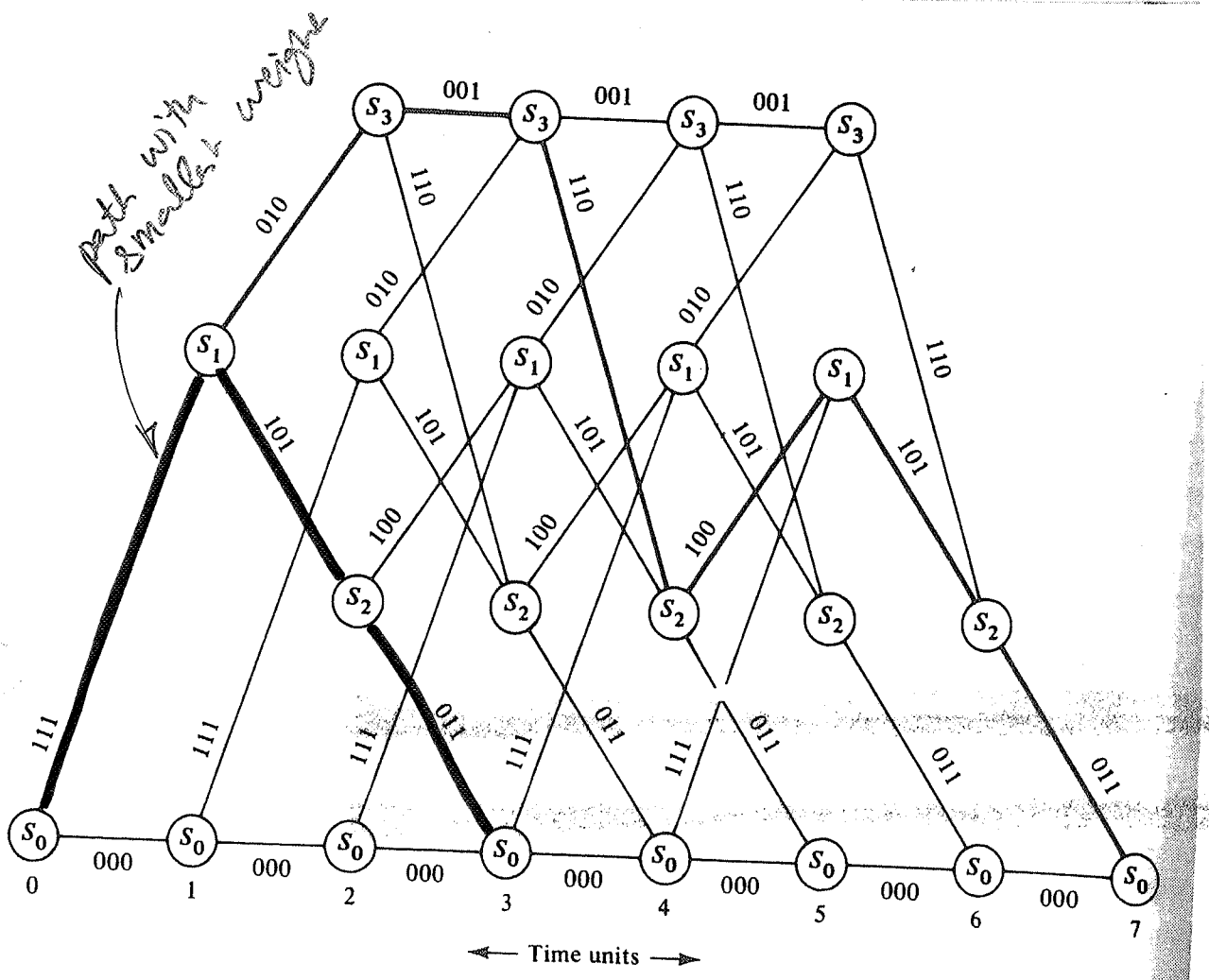


Figure 11.1 Trellis diagram for a (3, 1, 2) code with $L = 5$.

Example

- Assume that the incorrect path is the weight 7 path
- A <u>first error event</u> is made if, in the 7 positions in which the correct & incorrect paths differ, the received sequence agrees with incorrect path in 4 or more locations.
  (Remember the channel here is BSC, so its output is either 0 or 1)

$$P_7 = P[\text{4 or more 1's in seven positions}]$$
$$= \sum_{e=4}^{7} \binom{7}{e} p^e (1-p)^{7-e}$$

- If weight 8 path is the incorrect one, then

$$P_8 = \frac{1}{2}\binom{8}{4} p^4 (1-p)^4 + \sum_{e=5}^{8} \binom{8}{e} p^e (1-p)^{8-e}$$

↑ We have a tie. decide at random

In general, if the incorrect path has weight d, a <u>first event error</u> is made with probability

$$P_d = \begin{cases} \sum_{e=\frac{d+1}{2}}^{d} \binom{d}{e} p^e (1-p)^{d-e} & (\text{odd } d) \\ \frac{1}{2}\binom{d}{d/2} p^{d/2}(1-p)^{d/2} + \sum_{e=\frac{d}{2}+1}^{d} \binom{d}{e} p^e (1-p)^{d-e} & (\text{even } d) \end{cases}$$

- Since all non-zero paths of length j or less can cause a first even error at time j, the <u>first event error probability at time j</u>, $P_f(E,j)$ can be upper bounded by the sum of the error probabilities of each of the paths.
- If all paths of length greater than j are also included in the bound, we obtain

$$P_f(E,j) < P_7 + P_8 + P_9 + 2P_{10} + \cdots$$

$$= \sum_{d=d_{free}}^{\infty} a_d P_d$$

a_d ↑ # of codewords with weight j

- Since this bound is independent of j, the first event error probability, $P_f(E)$, at any time instant can be upper bounded by

$$P_f(E) < \sum_{d=d_{free}}^{\infty} a_d P_d$$

- For odd values of d

$$P_d = \sum_{e=\frac{d+1}{2}}^{d} \binom{d}{e} p^e (1-p)^{d-e}$$

$$< \sum_{e=\frac{d+1}{2}}^{d} \binom{d}{e} p^{d/2} (1-p)^{d/2} = p^{d/2}(1-p)^{d/2} \sum_{e=\frac{d+1}{2}}^{d} \binom{d}{e}$$

(p<1)

$$< p^{d/2}(1-p)^{d/2} \sum_{e=0}^{d} \binom{d}{e} = 2^d p^{d/2} (1-p)^{d/2}$$

$$= \left(2\sqrt{p(1-p)}\right)^d$$

- Same bound also holds for even d
- Hence

$$P_f(E) < \sum_{d=d_{free}}^{\infty} a_d \left(2\sqrt{p(1-p)}\right)^d$$

$$= T(\delta)\Big|_{\delta = 2\sqrt{p(1-p)}}$$

- Note : The final path can diverge from and remerge with correct path any number of times, i.e., it can contain any number of event errors.

- Key point : Event error probability ($P(E)$) can be upper bounded by the first event error probability i.e.

$$P(E) < P_f(E) < \sum_{d=d_{free}}^{\infty} a_d P_d = T(\delta)\Big|_{\delta=\left(2\sqrt{p(1-p)}\right)}$$

for BSC channels

- For small p, the bound is dominated by its first term

$$\Rightarrow P(E) \approx a_{d_{free}} 2^{d_{free}} p^{d_{free}/2}$$

Hence $a_{d_{free}}$ & d_{free} are critical in determining $P(E)$.

- For a binary input, additive white Gaussian noise (AWGN)

$$P(\mathcal{E}) < T(\delta)\Big|_{\delta = e^{-R_c E_b/N_0}}$$

R_c: code rate

E_b: energy per information bit ($\frac{E}{R}$ = energy per symbol)

N_0: one-side noise power spectral density

- For large E_b/N_0

$$P(\mathcal{E}) \simeq a_{d_{free}} e^{-R_c d_{free} E_b/N_0}$$

The event error probability bound can be modified to find a bound on <u>bit error probability</u> $P_b(\mathcal{E})$, the expected # of info bits in error per unit time.

- Each event error causes a # of info bit errors equal to the # of non-zero info bits on the incorrect path

- Hence if each term P_d in the event error probability is weighted by the total # of non-zero info bits on all weighted paths, a bit error bound is obtained

$$P_b(E) < \sum_{d=d_{free}}^{\infty}{}' B_d P_d$$

B_d: total # non-zero info bits on all weight d paths

- Since $T(\delta, N) = \sum_d \sum_b{}' A_{d,b} \delta^d N^b$

$$\left.\frac{\partial T(\delta,N)}{\partial N}\right|_{N=1} = \sum_d \sum_b{}' b A_{d,b} \delta^d = \sum_d B_d \delta^d$$

where $B_d = \sum_b{}' b A_{d,b}$

- For BSC

$$P_b(E) < \left.\frac{\partial T(\delta,N)}{\partial N}\right|_{\delta = 2\sqrt{p(1-p)},\ N=1}$$

$$\approx B_{d_{free}} (4p)^{d_{free}/2} \quad (\text{for small } p)$$

- For a binary input, AWGN channel

$$P_b(E) < \left. \frac{\partial T(\delta, N)}{\partial N} \right|_{\substack{\delta = e^{-R_c E_b/N_0} \\ N=1}}$$

$$\approx b_{d_{free}} e^{-R d_{free} E_b/N_0} \quad \text{(for large } E_b/N_0\text{)}$$

that is non-zero in the first time unit, i.e., the information sequence for a detour beginning at node 1 when $\underline{0}$ is the correct path. The following result now follows from (6.24) and our previous observations about finite trellises.

Lemma 6.2: For any correct path and any $j, a(d,i)$ is the number of detours beginning at node j that have Hamming distance d from the corresponding segment of the correct path and contain i information bit errors in the $L_t = \infty$ trellis for the given convolutional encoder, and is an upper bound on this number for every finite L_t.

6.5 A Tight Upper Bound on the Bit Error Probability of a Viterbi Decoder

We will shortly use $T(D,I)$ as the basis for a useful upper bound on the bit error probability, P_b, of a Viterbi decoder for a binary convolutional code. First, however, we recall and extend our understanding of two matters that arose in our discussion of block codes.

First, we recall the Bhattacharyya bound (5.28) for a block code with two codewords. We note that for a binary-input DMC, the summation on the right side of (5.28) equals 1 for all n where $x_{1n} = x_{2n}$ but equals

$$\sum_y \sqrt{P_{Y|X}(y|0)P_{Y|X}(y|1)} = 2^{-D_B}$$

for those n where $x_{1n} \neq x_{2n}$. But since the number of these latter n is just the Hamming distance, $d(\underline{x}_1, \underline{x}_2)$, between the two codewords, we see that (5.28) implies:

Lemma 6.3: For a binary-input DMC and the code $(\underline{x}_1, \underline{x}_2)$, the worst-case block error probability for an ML decoder satisfies

$$(P_B)_{wc} \leq (2^{-D_B})^{d(\underline{x}_1, \underline{x}_2)} \tag{6.25}$$

where D_B is the Bhattacharyya distance between the two channel input digits and $d(\underline{x}_1, \underline{x}_2)$ is the Hamming distance between the two codewords.

Next, we recall the definition (4.46) of the bit error probability of a decoder for a block code with K information bits, namely,

$$P_b = \frac{1}{K}\sum_{i=1}^{K} P_{e,i}$$

where $P_{e,i}$ is the probability of a decoding error in the i-th information bit. We now define V_i to be the indicator random variable for an error in the i-th information bit, i.e., $V_i = 1$ when such an error occurs and $V_i = 0$ otherwise. But then

$$P_{V_i}(1) = P_{e,i}$$

so that

$$\mathrm{E}[V_i] = P_{e,i}\,. \tag{6.26}$$

Using (6.26) in (4.46), we find

$$\begin{aligned} P_b &= \frac{1}{K}\sum_{i=1}^{K} \mathrm{E}[V_i] \\ &= \frac{1}{K}\mathrm{E}\left[\sum_{i=1}^{K} V_i\right] \\ &= \mathrm{E}\left[\frac{1}{K}\sum_{i=1}^{K} V_i\right]. \end{aligned} \tag{6.27}$$

But the random variable $\sum_{i=1}^{K} V_i$ is the total number of information bit errors made by the decoder so (6.27) can be stated as:

Lemma 6.4: The bit error probability of a decoder for a block code equals the average fraction of erroneously decoded information bits.

We are now ready to derive a tight upper bound on P_b for a Viterbi decoder for the trellis code of length $L_t + T$ branches generated by a convolutional encoder. We begin by defining the random variable W_j as the number of information bit errors made by the Viterbi decoder as it follows a detour beginning at node j. Naturally, $W_j = 0$ when the decoder does not follow a detour beginning at node j either because it is already on a detour that began earlier or because it follows the correct path from node j to node $j+1$. Then $\sum_{j=1}^{L_t} W_j$ is the total number of information bit errors made by the Viterbi decoder so that Lemma 6.4 gives

$$P_b = \mathrm{E}\left[\frac{1}{k_0 L_t}\sum_{j=1}^{L_t} W_j\right], \tag{6.28}$$

where k_0 is the *number of information bits per time unit* that enter the convolutional encoder. For the encoder of Figure 6.1, $k_0 = 1$ – but of course we wish our treatment here to be general.

Next, we define B_{jk} to be the event that the Viterbi decoder follows the k-th detour at node j on the correct path (where the detours are numbered in any order). Letting d_k be the Hamming distance between this detour and the corresponding segment of the correct path, we can invoke Lemma 6.3 to assert that

$$P(B_{jk}) \leq (2^{-D_B})^{d_k}. \tag{6.29}$$

Letting i_k be the number of information bit errors on this detour, we see that

$$W_j = \begin{cases} i_k, & \text{if } B_{jk} \text{ occurs for some } k \\ 0, & \text{otherwise.} \end{cases} \tag{6.30}$$

Thus, (6.30) and the theorem on total expectation imply

$$\mathrm{E}[W_j] = \sum_k i_k P(B_{jk}). \tag{6.31}$$

Using (6.29) in (6.31) now gives

$$\mathrm{E}[W_j] \leq \sum_k i_k \left(2^{-D_B}\right)^{d_k}. \tag{6.32}$$

But, by the definition of $a_j(d,i)$, the number of indices k in (6.32) for which $i_k = i$ and $d_k = d$ is just $a_j(d,i)$ so that (6.32) can be written

$$\mathrm{E}[W_j] \leq \sum_i \sum_d i a_j(d,i) \left(2^{-D_B}\right)^d. \tag{6.33}$$

But Lemma 6.2 states that

$$a_j(d,i) \leq a(d,i) \tag{6.34}$$

for all j. Using (6.34) in (6.33) and then (6.33) in (6.28), we have finally

$$P_b \leq \frac{1}{k_0}\sum_i \sum_d i a(d,i) \left(2^{-D_B}\right)^d, \tag{6.35}$$

which is our desired bound. Notice that (6.35) holds for every L_t – even for $L_t = \infty$ when we never insert the tail of dummy information bits to return the encoder to the zero state and when R_t is the true information rate of the trellis code.

It remains only to express the right side of (6.35) in terms of the transmission gain $T(D, I)$ of the detour flowgraph. Differentiating with respect to I in (6.19) gives

$$\frac{\partial T(D,I)}{\partial I} = \sum_i \sum_d i a(d,i) I^{i-1} D^d. \tag{6.36}$$

Comparing (6.35) and (6.36), we obtain the following result:

Theorem 6.1: The bit error probability for Viterbi decoding of the trellis code generated by a binary convolutional encoder used on a DMC satisfies

$$P_b \leq \frac{1}{k_0} \frac{\partial T(D,I)}{\partial I}\bigg|_{I=1, D=2^{-D_B}} \tag{6.37}$$

provided that the power series on the right of (6.36) converges at $I = 1$ and $D = 2^{-D_B}$, where k_0 is the number of information bits per time unit that enter the convolutional encoder, where $T(D, I)$ is the transmission gain of the detour flowgraph of the convolutional encoder, and where D_B is the Bhattacharyya distance between the two input letters of the DMC.

The bound (6.37) holds for all trellis lengths L_t (including $L_t = \infty$) and all assignments of probabilities to the $k_0 \cdot L_t$ "information bits".

For the encoder of Figure 6.1, we find by differentiation in (6.20) that

$$\frac{\partial T(D,I)}{\partial I} = \frac{D^5}{(1-2ID)^2}.$$

Suppose this convolutional code is used on a BEC with erasure probability δ. Then (5.34) gives

$$2^{-D_B} = \delta$$

so that (6.37) becomes (because $k_0 = 1$ for this encoder)

$$P_b \leq \frac{\delta^5}{(1-2\delta)^2}.$$

In particular, for $\delta = \frac{1}{10}$, our bound becomes

$$P_b \leq 1.56 \times 10^{-5}.$$

It has been determined from simulations that the bound of (6.37) is so tight in general that it can be used to design convolutional decoders for particular applications without appreciable extra complexity resulting from the fact that an upper bound on P_b, rather than the true P_b, is used. This should not surprise us in light of the tightness of the Bhattacharyya bound on which (6.37) is based.

When the memory, T, of the convolutional encoder is very small, it is practical to calculate $T(D, I)$, as we have done here for the encoder of Figure 6.1. For larger values of T but still in the range where Viterbi decoding is practical (say, $k_0 T \leq 10$), it is impractical to calculate $T(D, I)$ as a function of D and I from the detour flowchart, although it is still easy enough to calculate $T(D, I)$ for particular values of D and I. In this case, we can still make use of the bound (6.37) by approximating the derivative, for instance as

$$\frac{\partial T(D,I)}{\partial I}\bigg|_{I=1, D=2^{-D_B}} \approx \frac{T(2^{-D_B}, 1.01) - T(2^{-D_B}, .99)}{.02} \tag{6.38}$$

which requires the evaluation of $T(D, I)$ for only two particular cases.

Finally, we should mention that when $L_t = \infty$ (as is normally the case for practical systems that use Viterbi decoders) we do not want, of course, to wait until the end of the trellis before announcing the decoded information sequence. Theoretical considerations indicate, and simulations have confirmed, that P_b is virtually unchanged when the Viterbi decoder is forced to make its decisions with a delay of about 5 constraint lengths. In this case, the decoded information bits at time unit j are taken as their values in the information sequence for the currently best path to the state with the highest metric at time unit $j + 5(T + 1)$.

6.6 Random Coding – Trellis Codes and the Viterbi Exponent

We are about to develop a random coding bound for trellis codes which will show that, in an important way, trellis codes are superior to ordinary block codes. To do this, however, we must first suitably generalise our concept of a "trellis code".

The general form of what we shall call an "(n_0, k_0) trellis encoder with memory T" is shown in Figure 6.10. At each "time unit", k_0 information bits enter the encoder while n_0 channel input digits leave the encoder. Thus, the *rate* of the trellis code is

$$R_t = \frac{k_0}{n_0} \text{ bits/use} \tag{6.39}$$

(which is of course a true information rate only if all the information bits are statistically independent and each is equally likely to be 0 or 1.) Notice that, although our information symbols are binary, the encoded symbols are digits in the channel input alphabet. The n_0 encoded digits formed each time unit depend only on the current k_0 information bits and on the $k_0 T$ information bits in the T most recent time units, but the functional form of this dependence is arbitrary – in particular this function need not be linear nor time-invariant as it was for the convolutional codes previously considered. By way of convention, we assume that the "past" information bits are all zeroes when encoding begins.

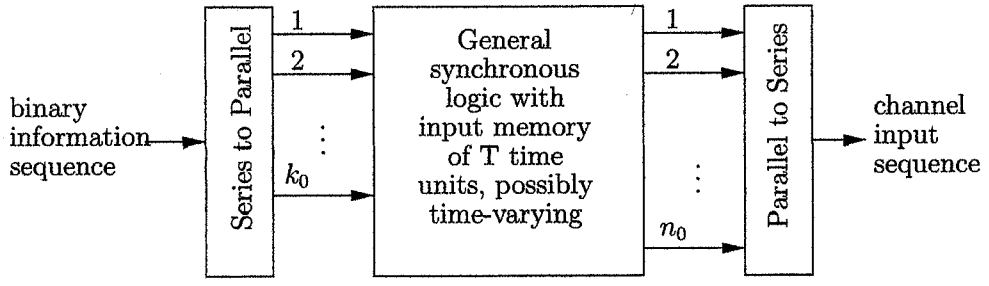


Figure 6.10: A general (n_0, k_0) trellis encoder with memory T and rate $R_t = \frac{k_0}{n_0}$.

It makes the notation much simpler if we write

$$\underline{U}_i = [U_{(i-1)k_0+1}, U_{(i-1)k_0+2}, \ldots, U_{ik_0}] \tag{6.40}$$

and

$$\underline{X}_i = [X_{(i-1)n_0+1}, X_{(i-1)n_0+2}, \ldots, X_{in_0}] \tag{6.41}$$

for the "byte" of k_0 information bits that enter the encoder and the "byte" of n_0 channel input symbols that leave the encoder, respectively, at time unit $i = 1, 2, 3, \ldots$. Then Figure 6.10 corresponds to the functional relationship

$$\underline{X}_i = f_i(\underline{U}_i, \underline{U}_{i-1}, \ldots, \underline{U}_{i-T}) \tag{6.42}$$

where, by way of convention,

$$\underline{U}_j = [0, 0, \ldots, 0], \qquad j < 1. \tag{6.43}$$

Oct 31

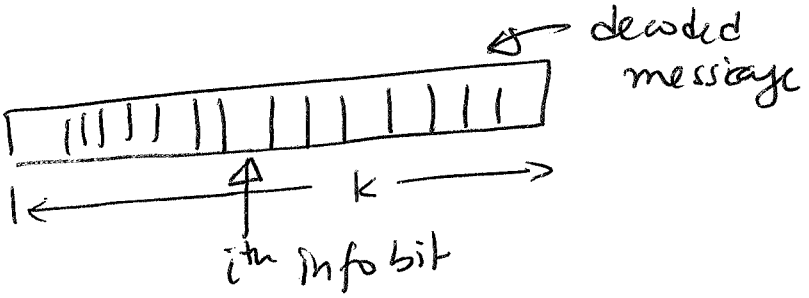
BIT ERROR PROBABILITY OF A VITERBI DECODER

LET'S look at Block codes first.

Bit error probability with K information bits

$$P_b = \frac{1}{K} \sum_{i=1}^{K} P_{e,i} \quad \text{---} \quad (*)$$

$P_{e,i}$ = prob of decoding error in the i^{th} information bit



Let's rewrite (*)

Define $V_i = \begin{cases} 1 & \text{if } i^{th} \text{ decoded bit in error} \\ 0 & \text{if not in error} \end{cases}$

V_i is a random variable

& $P_{V_i}(V_i = 1) = P_{e,i}$

OR $E(V_i) = 0 \cdot P(V_i = 0) + 1 \cdot P(V_i = 1)$

$= P_{e,i}$

Then, bit error probability

$$P_b = \frac{1}{K} \sum_{i=1}^{K} \mathbb{E}(V_i)$$

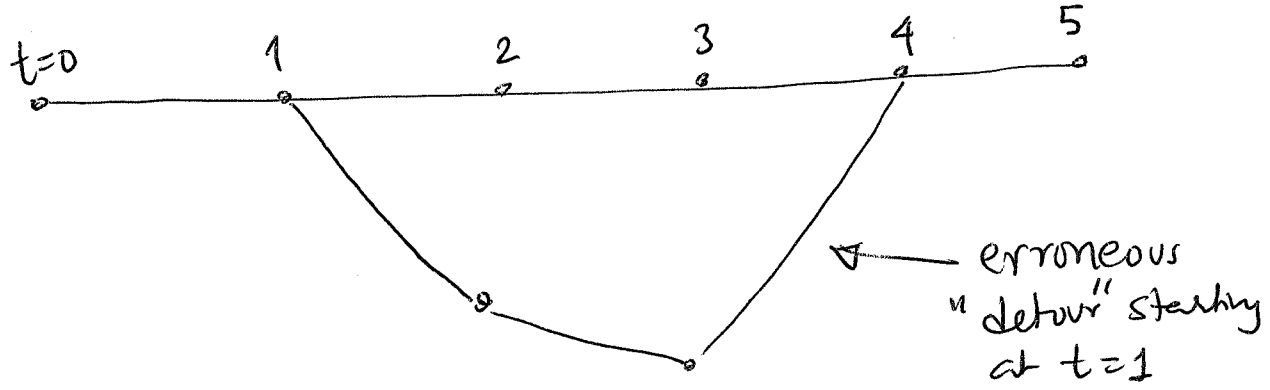
$$= \sum_{i=1}^{K} \frac{1}{K} \mathbb{E}(V_i)$$

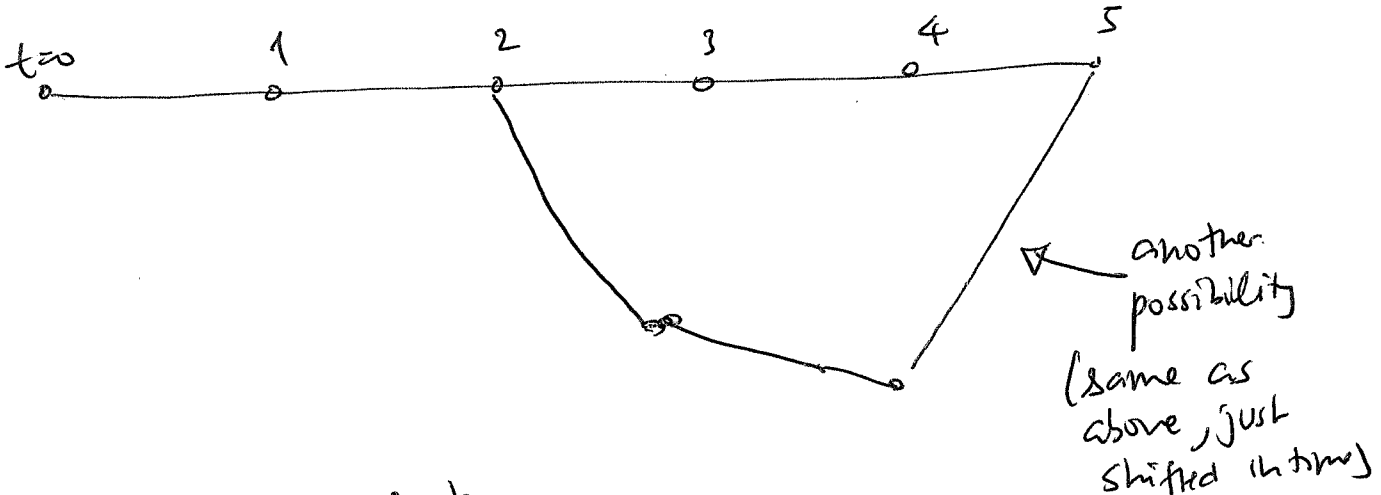
$$= \mathbb{E}\left[\frac{1}{K} \sum_{i=1}^{K} V_i\right]$$

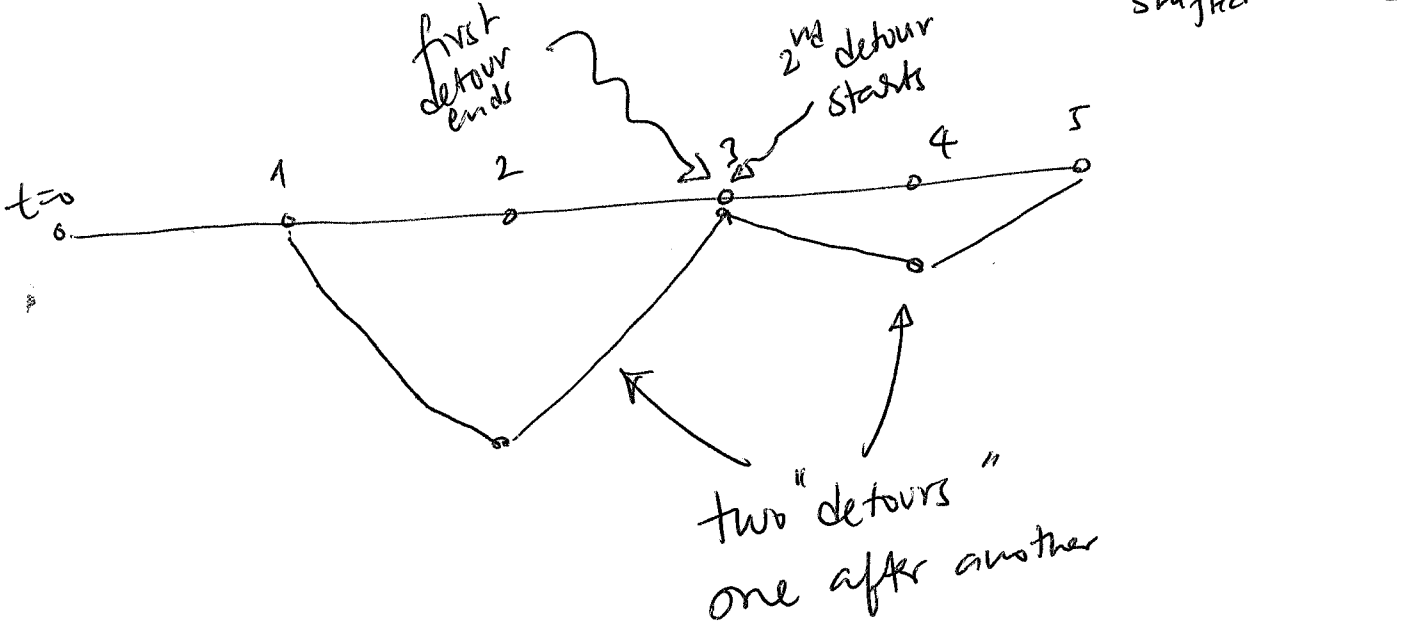
Random variable $\sum_{i=1}^{K} V_i$ = total # of errors

In other words, P_b = Average fraction of bits in error

Assume all zero codeword was sent







At any time, while decoding, you can be only one possible detour if making an error (following the wrong path).

An error event is when you give up the right path for the wrong one.

So let's treat the convolutional code like a block code by looking at finite # of info bits followed by m flushing 0's

Let L = # of information bits

total # of branches/stages in encoding

$= L + m$

Define

W_j = total # of errors made on a "detour"/path starting at node j ($t = j$)

Random Variable

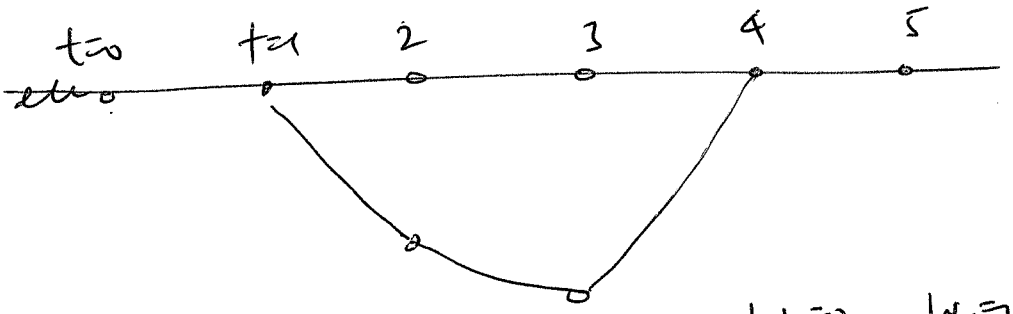
Now $W_j = \begin{cases} 0, & \text{if no path starts at } j \\ \#\text{ of 1's in path } k, & \text{if path } k \text{ is chosen} \end{cases}$

total of 1's in the input (not the codeword)

Why $W_j = 0$ sometimes?

* it is following the correct path "
* it is already on another incorrect path "

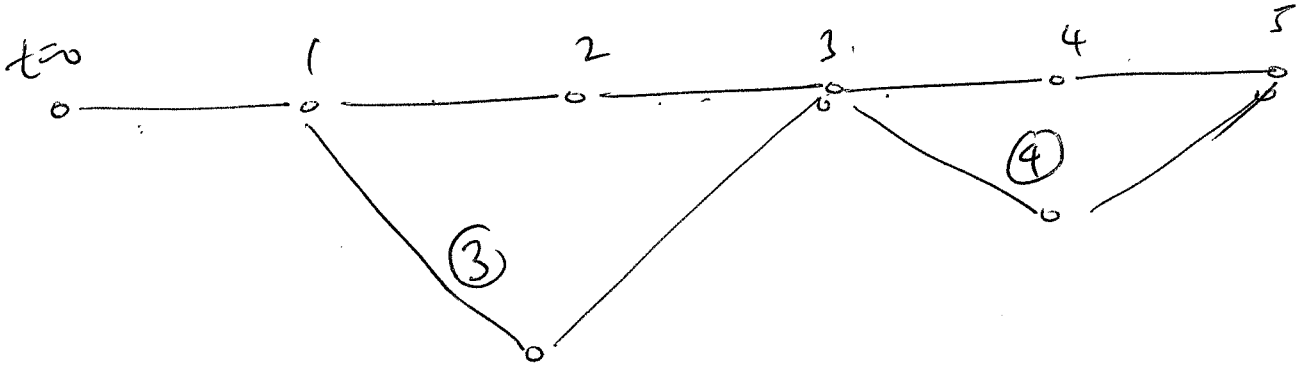
Example



$W_0=0 \quad W_1=7 \quad W_2=0 \quad W_3=0 \quad W_4=0 \quad W_5=0$

Total # of errors $= 7 = \sum_{j=0}^{5} W_j$

Another example



$w_0 = 0 \quad w_1 = 3 \quad w_2 = 0 \quad w_3 = 4 \quad w_4 = 0 \quad w_5 = 0$

total # of errors (in info bits not codeword bits) $= \sum_{j=0}^{5} w_j = 7$

So Total # of errors $= \sum_{j=1}^{L} w_j$ ← not L+m (last m bits have to be zero)

$$\Rightarrow P_b = \mathbb{E}\left[\frac{1}{Lk}\sum_{j=1}^{L} w_j\right]$$

$$= \frac{1}{Lk}\sum_{j=1}^{L} \mathbb{E}[w_j]$$

↑ Expected # of errors in decoded information bits for error paths starting at time $t=j$

Recall W_j is defined as # of errors initiated at time j, which is non-zero only if a new path started at time j.

Let B_{jk} be the path which starts at time j if an error path is started at all. Then W_j is

$$W_j = \begin{cases} 0 & \text{if correct path is followed or another error event is in progress} \\ i_k & \text{If error event } B_{jk} \text{ is started} \end{cases}$$

$$E[W_j] = \sum_k 0 \cdot [1 - P(B_{jk})] + i_k P(B_{jk})$$

↑ summation over all possible error events

Remember error events start @ zero state and finish @ zero state & never visit zero state in their "journey"

$$E[W_j] = \sum_i \sum_d i \, a_{d,i} \, P_d$$

- P_d ← probability of choosing a path with weight d
- $a_{d,i}$ ← # of paths with weight d & i bits in error
- d ← weight of path
- $a_{d,i}$: weight distribution (generating function)

Last time, we bounded P_d

$$P_d \leq \left[2\sqrt{p(1-p)}\right]^d$$

(independent of i)

$$\Rightarrow \mathbb{E}[W_j] \leq \sum_i \sum_d i\, a_{d,i} \left[2\sqrt{p(1-p)}\right]^d$$

$$\Rightarrow P_b \leq \frac{1}{k}\left[\frac{1}{L}\sum_{j=1}^{L} \mathbb{E}[W_j]\right]$$

$$= \frac{1}{k}\,\mathbb{E}(W_j) \qquad \left[\text{since all } \mathbb{E}(W_j) \text{ are same}\right]$$

Now

$$T(\delta, N) = \sum_i \sum_d a_{d,i}\, \delta^d N^i$$

- $a_{d,i}$: # of paths with para (d,i)
- δ^d: wt of path
- N^i: # of 1's in the info stream

$$\frac{\partial}{\partial N} T(\delta, N) = \sum_i \sum_d i\, a_{d,i}\, N^{i-1} \delta^d$$

$$\Rightarrow \boxed{P_b \leq \frac{1}{k}\left[\frac{\partial}{\partial N} T(\delta, N)\right]_{\substack{\delta = 2\sqrt{p(1-p)} \\ N=1}}}$$