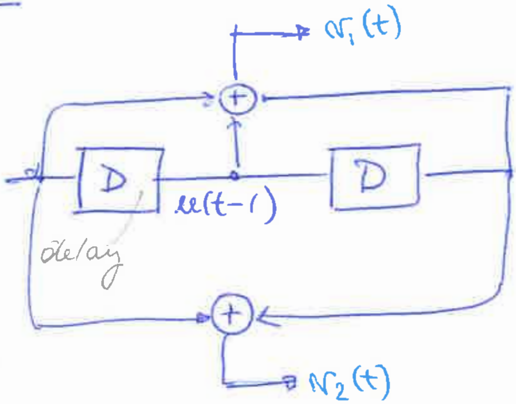
Convolutional Codes

Tree codes :

Example 1 input: $u(t)$



$$v_1(t) = u(t) + u(t-1) + u(t-2)$$

$$v_2(t) = u(t) + u(t-2)$$

$$u(t) = (u(t=0) \quad u(1) \quad h(2) \ \ldots\ldots)$$

$$v_1(t) = (v_1(t=0) \ \ldots)$$

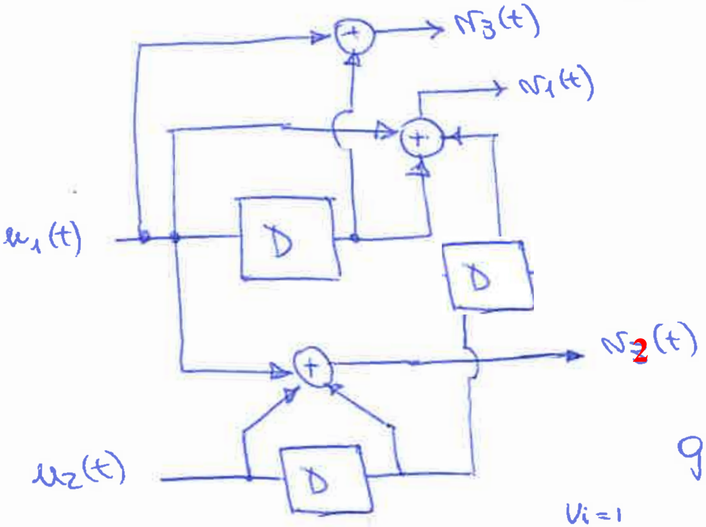
$$v_2(t) = (v_2(t=0 \quad v_2(t=1) \ \ldots)$$

General $k \times m$: $\frac{k}{m}$ convolutional code (k inputs, m outputs)

$$v_j(t) = \sum_{i=1}^{k} \sum_{m=0}^{\nu_i} g_{ij}(m)\, u_i(t-m)$$

Example 2

in the previous example $\left\{ \begin{array}{l} g_{\substack{i=1 \\ j=2}}(m=1) = 0 \\ \nu_i = 2 \end{array} \right.$



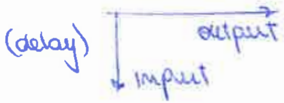
$$v_1(t) = u_1(t) + u_1(t-1) + u_2(t-2)$$

$$v_2(t) = u_2(t) + u_2(t-1) + u_1(t)$$

$$v_3(t) = u_1(t) + u_1(t-1)$$

general expression :

$$v_1 = \sum_{m=0}^{\nu_1 = 1} g_{11}(m)\, u_1(t-m) + \sum_{m=0}^{\nu_2 = 2} g_{21}(m)\, u_2(t-m)$$



$$g_{ij}(0) = \overset{D^0}{\begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 0 \end{bmatrix}}_{2\times 3} \quad g_{ij}(1) = \overset{D^1}{\begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}}_{2\times 3} \quad g_{ij}(2) = \overset{D^2}{\begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix}}_{2\times 3}$$

transition to polynomials

Conversion of previous matrices to polynomials:

$$\begin{bmatrix} 1+D & 1 & 1+D \\ D^2 & 1+D & 0 \end{bmatrix}$$

will see soon →

(PGM)

Formal Power Series (FPS)

$$v_j(D) \triangleq \sum_{t=0}^{\infty} v_j(t) D^t$$

$$u_i(D) \triangleq \sum_{t=0}^{\infty} u_i(t) D^t$$

$$v_j(D) = \sum_{t=0}^{\infty} \underbrace{\sum_{i=1}^{k} \sum_{m=0}^{v_i} g_{ij}(m) \cdot u_i(t-m)}_{v_j(t)} D^t =$$

$$= \sum_{t=0}^{\infty} \sum_{i=1}^{k} \sum_{m=0}^{v_i} g_{ij}(m)\, u_i(t-m)\, D^m D^{t-m}$$

$$= \sum_{i=1}^{k} \sum_{m=0}^{v_i} g_{ij}(m) D^m \sum_{t=0}^{\infty} u_i(t-m) D^{t-m}$$

- $u_i(t) \quad t<0$ $\qquad \sum_{i=1}^{k} \sum_{m=0}^{v_i} g_{ij}(m) D^m \sum_{t=m}^{\infty} u_i(t-m) D^{t-m}$

Change rows

$$r_j(D) = \sum_{i=1}^{K} \underbrace{\sum_{m=0}^{\nu_i} g_{ij}(m) D^m}_{g_{ij}(D)} \sum_{t=0}^{\infty} u_i(t) D^t$$

$$g_{ij}(D) \triangleq \sum_{m=0}^{\nu_i} g_{ij}(m) D^m$$

$$r_j(D) = \sum_{i=1}^{K} g_{ij}(D) u_i(D)$$

$$g_{ij}(D) = g_{ij}(0) + g_{ij}(1) \cdot D + g_{ij}(2) \cdot D^2 + \dots \text{ up to } \nu_i$$

$$u_i(D) = u_i(0) + u_i(1) D + u_i(2) D^2 + \dots$$

Polynomial generating matrix (PGM)

$$\underbrace{\begin{bmatrix} r_1(D) & r_2(D) & \dots & r_n(D) \end{bmatrix}}_{1 \times n} = \underbrace{\begin{bmatrix} u_1(D) & u_2(D) & \dots & u_k(D) \end{bmatrix}}_{1 \times k} \overbrace{\begin{bmatrix} g_{11}(D) & \dots & g_{1n}(D) \\ \vdots & & \vdots \\ g_{k1}(D) & \dots & g_{kn}(D) \end{bmatrix}}^{\text{PGM}}$$

1st example $\left\{\begin{array}{l} k = 1 \\ n = 2 \end{array}\right.$ $\Rightarrow$ means that we have 1 input sequence and 2 output sequences (sequences!! not bits!!)

memory $= 2 = \nu_i = \nu$

v1(t)=u(t)+u(t-1)+u(t-2)
v2(t)=u(t)+u(t-2)

$$[v_1(D) \quad v_2(D)] = (u(D)) \Big[\underbrace{1+D+D^2}_{g_{11}(D)} \quad \underbrace{1+D^2}_{g_{12}(D)} \Big]$$

Total memory of the encoder is $\nu \triangleq \sum_{i=1}^{K} \nu_i$

In example 2 : $\left.\begin{array}{l} u_1(t-1) \longrightarrow \nu_1 = 1 \\ u_2(t-2) \longrightarrow \nu_2 = 2 \end{array}\right\} \Rightarrow \nu = \nu_1 + \nu_2 = 3$ ✓

State Diagram

State $u(t)$ $u(t-1)$

$u(t)$ ↓ 1st state

$u(t-1)$ ↓ 2nd state

"suppose I'm in state 0, 0"

↓ means

(00) $\begin{cases} u(t) = 0 \\ u(t-1) = 0 \end{cases}$

(01) $\begin{cases} u(t) = 0 \\ u(t-1) = 1 \end{cases}$

$$v1(t)=u(t)+u(t-1)+u(t-2)$$
$$v2(t)=u(t)+u(t-2)$$

"we are pushing bits at the input"

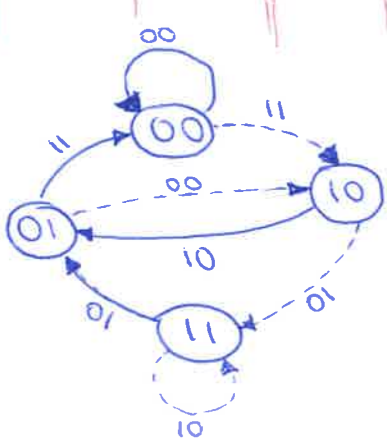
input	state		new state	new $u(t)$	$u(t-1)$	$u(t-2)$
→ 0	00	→	00	0	0	0
→ 0	01	→	00	0	0	1
- - → 1	00	→	10	1	0	0
- - → 1	01	→	10	1	0	1

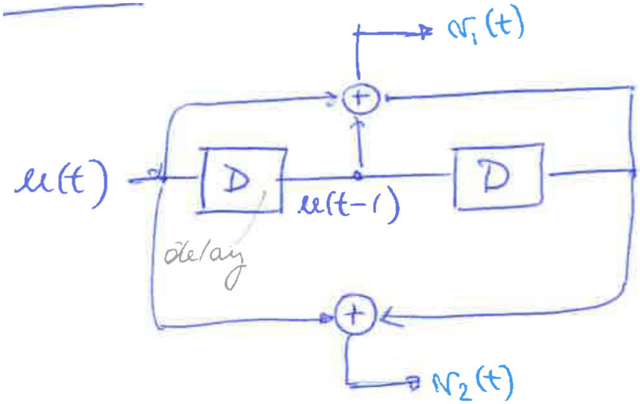
new state

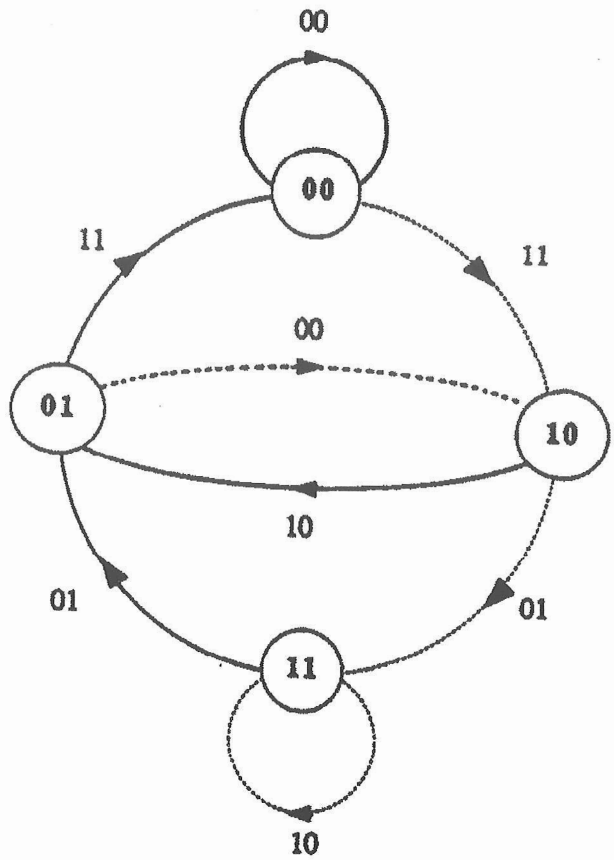
$\underbrace{u(t-1) \quad u(t-2)}_{\text{old state}}$

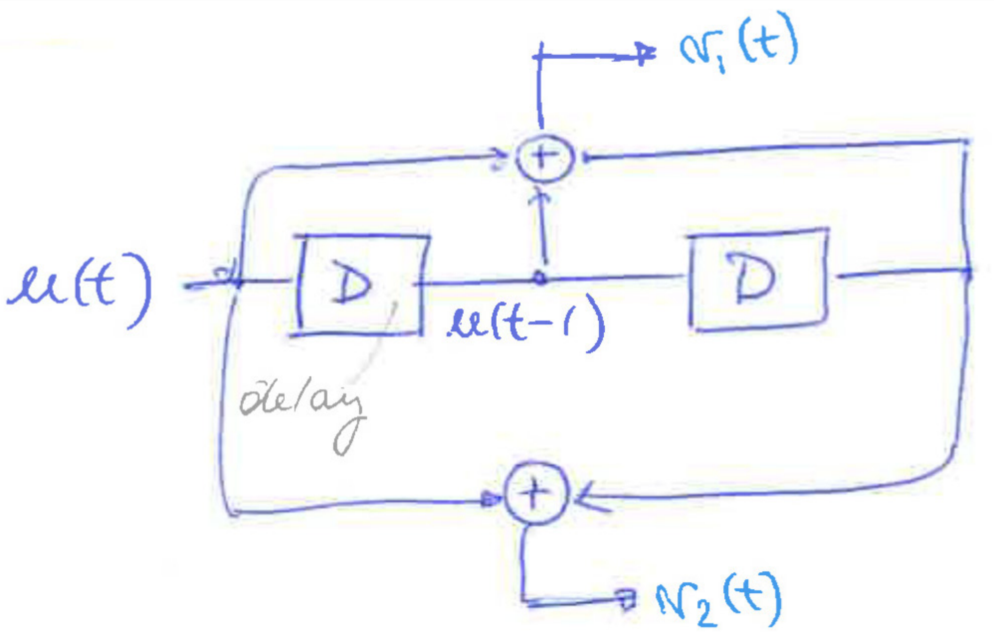
$u(t)$	$u(t-1)$	$u(t-2)$	$v_1(t)$	$v_2(t)$
0	0	0	0	0
0	0	1	1	1
1	0	0	1	1
1	0	1	0	0

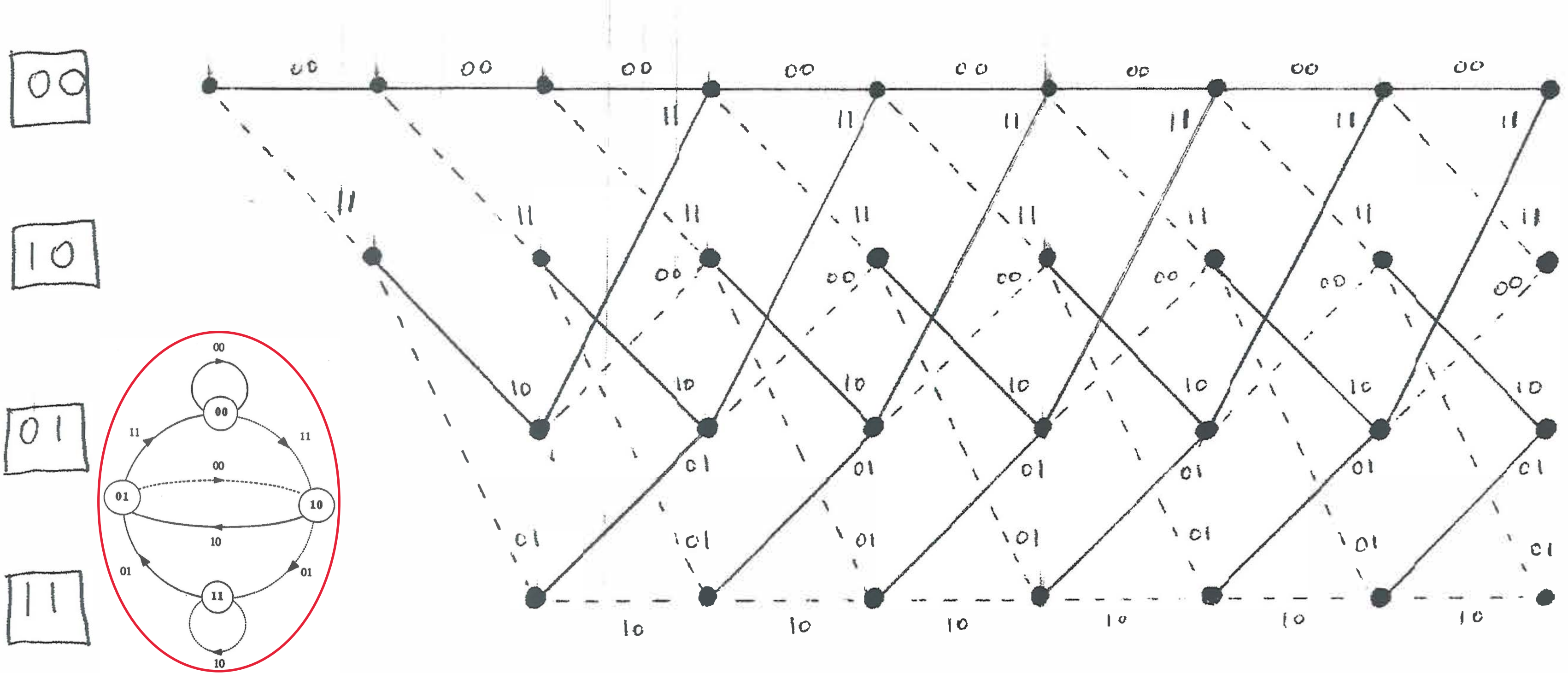
- The states represent the memory
- "what is kept in the register"



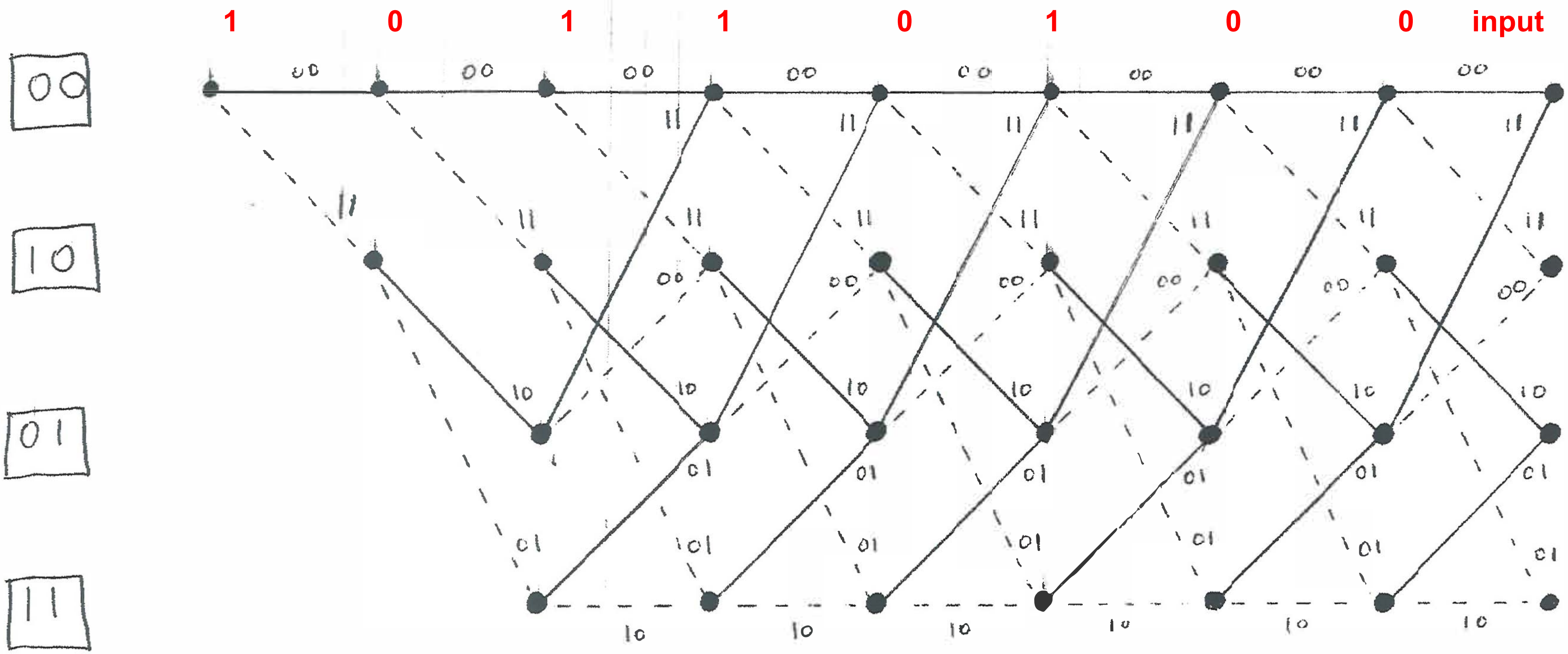



00
00
11
11
00
01
10
10
01
01
11
10

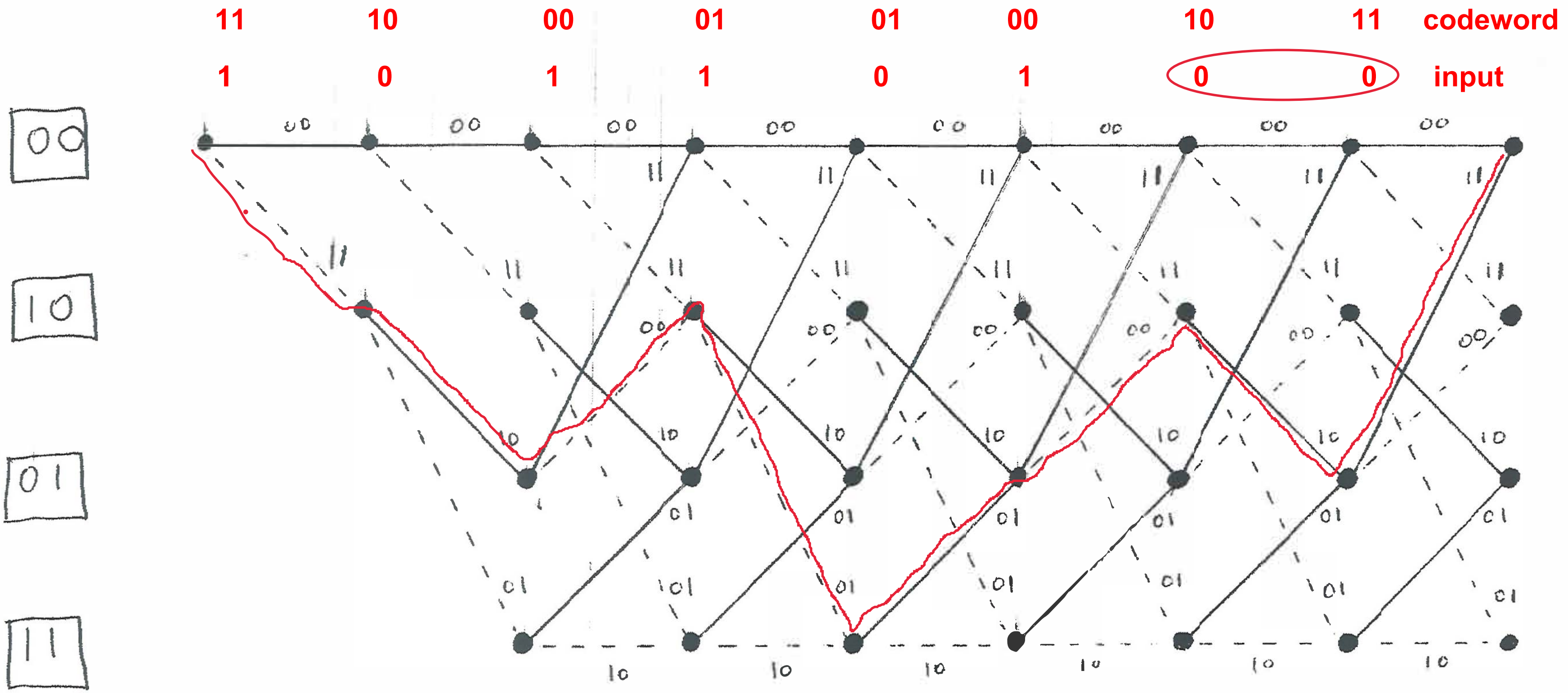

$v_1(t)$
$u(t)$
D
$u(t-1)$
D
delay
$v_2(t)$



00
10
01
11



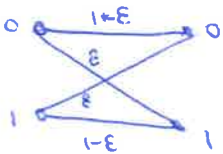
1
0
1
1
0
1
0
0
input
00
10
01
11



11 10 00 01 01 00 10 11 codeword
1 0 1 1 0 1 0 0 input
00
10
01
11

Errors

BSC



$\{\underline{u}(t)\}$

$\downarrow$

$\{\underline{v}(t)\} \leftarrow$ codeword

$\downarrow$ BSC

$\{\underline{r}(t)\}$

$\{v(t)\}$ $J = 1, 2, \dots, k$

$t = 0, 1, \dots, N-1 \longrightarrow$ in practice very long!

$\{r(t)\}$ $J = 1, \dots, k$

$t = 0, \dots, N-1$

You want to ~~correct~~ find $\underline{v} \in C$ s.t. maximize

P(detected is correct)

$$\max_{\underline{r} \in C} P\Big[\{r_i(t)\}_{\substack{i=1\to m \\ t=0\to N-1}} \,/\, \{v_i(t)\}_{\substack{t=1\to m \\ t=0\to N-1}}\Big] \longrightarrow \max \ln P\big[\{r_i(t)\} / \{v_i(t)\}\big]$$

NOTE: Assume independence in t & i

Linear combination of stat. indep. are still stat. indep!

$$\max \ln \prod_{t=0}^{N-1} \prod_{i=1}^{m} P\big(r_i(t)/v_i(t)\big) =$$

$$= \max \sum_{t=0}^{N-1} \sum_{i=1}^{m} \ln\Big[P\big(r_i(t)/v_i(t)\big)\Big]$$

$$P\big(r_i(t) \mid v_i(t)\big) = \varepsilon^d (1-\varepsilon)^{m-d}$$

$\varepsilon < 1$, we are penalizing more large Hamming distances

001
000
111

And then you add along the $t \to$ and you choose the path of min distance @ each node find path that is smaller. Add node + branch metric.

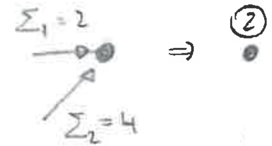
Viterbi Decoding

In previous example assume:

Input:	1	0	1	1	0	1	0	0
Output (Tx):	11	10	00	01	01	00	10	11
Received (Rx):	10	00	00	01	01	00	10	11
Decoded:	11	10	00	01	01	00	10	11

We complete the branches with the distances of the branches to the received vector

(x) number in each node is the sum through the branches

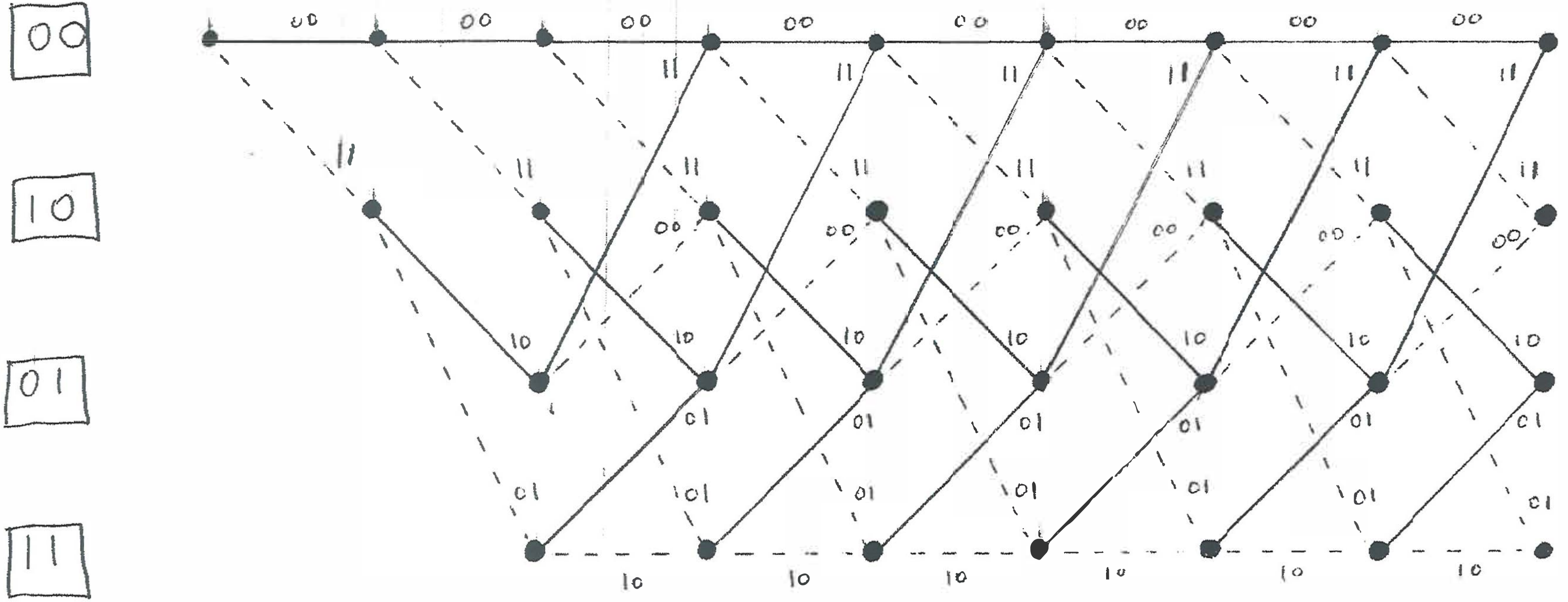
If 

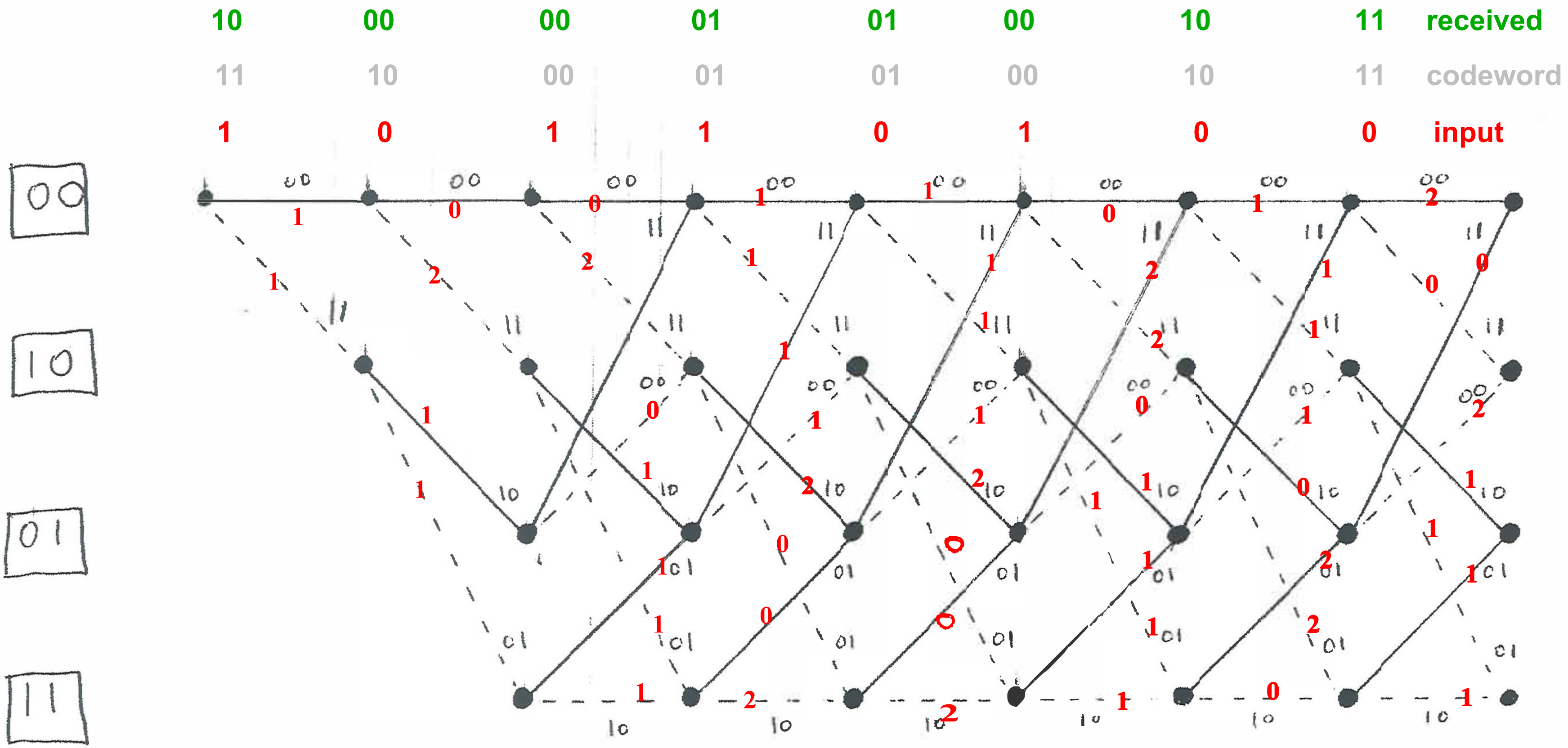
To decode:

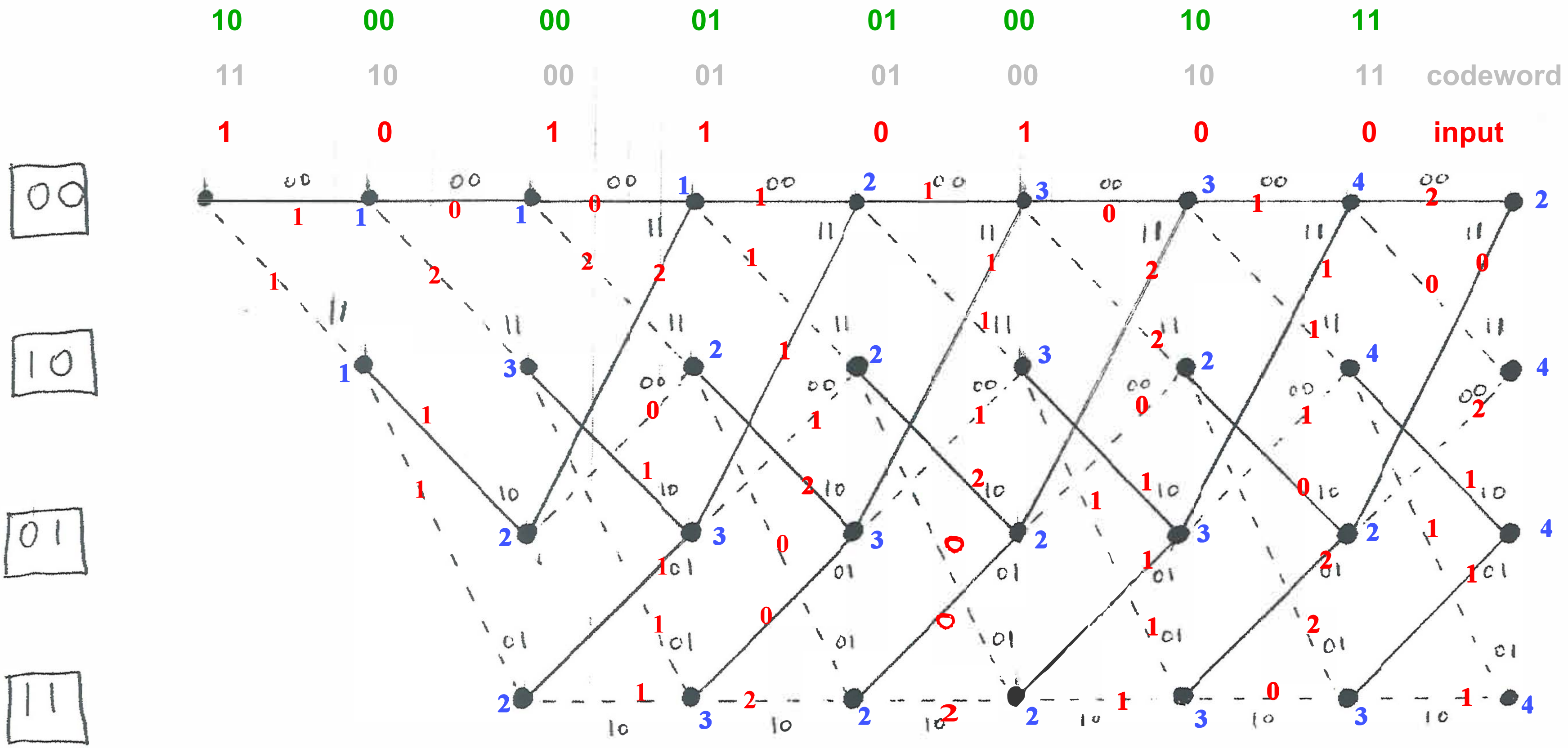
- Start by the last node with lowest weight.
- Follow the most probable path, that means following the branches with less weight

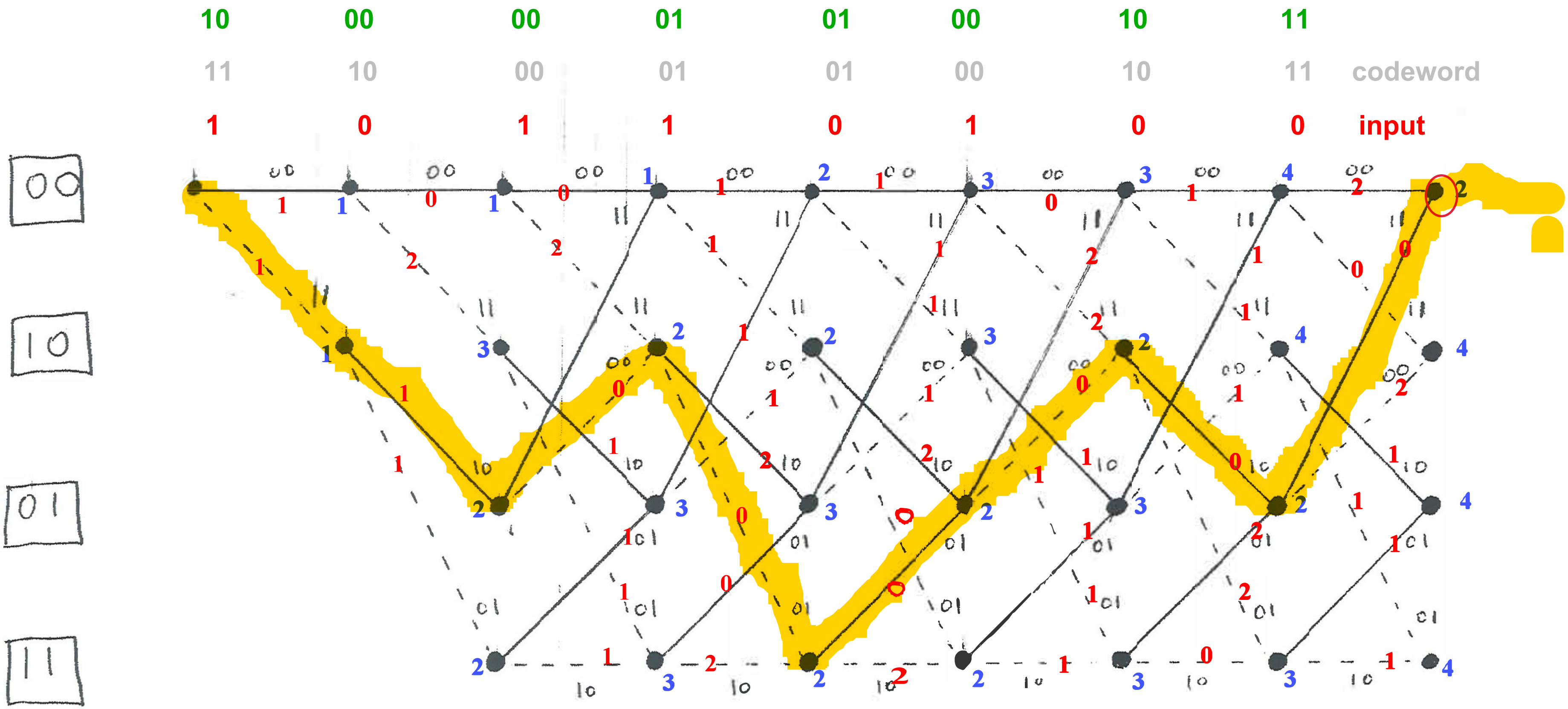
The error correcting capabilities of the code depend on its length.

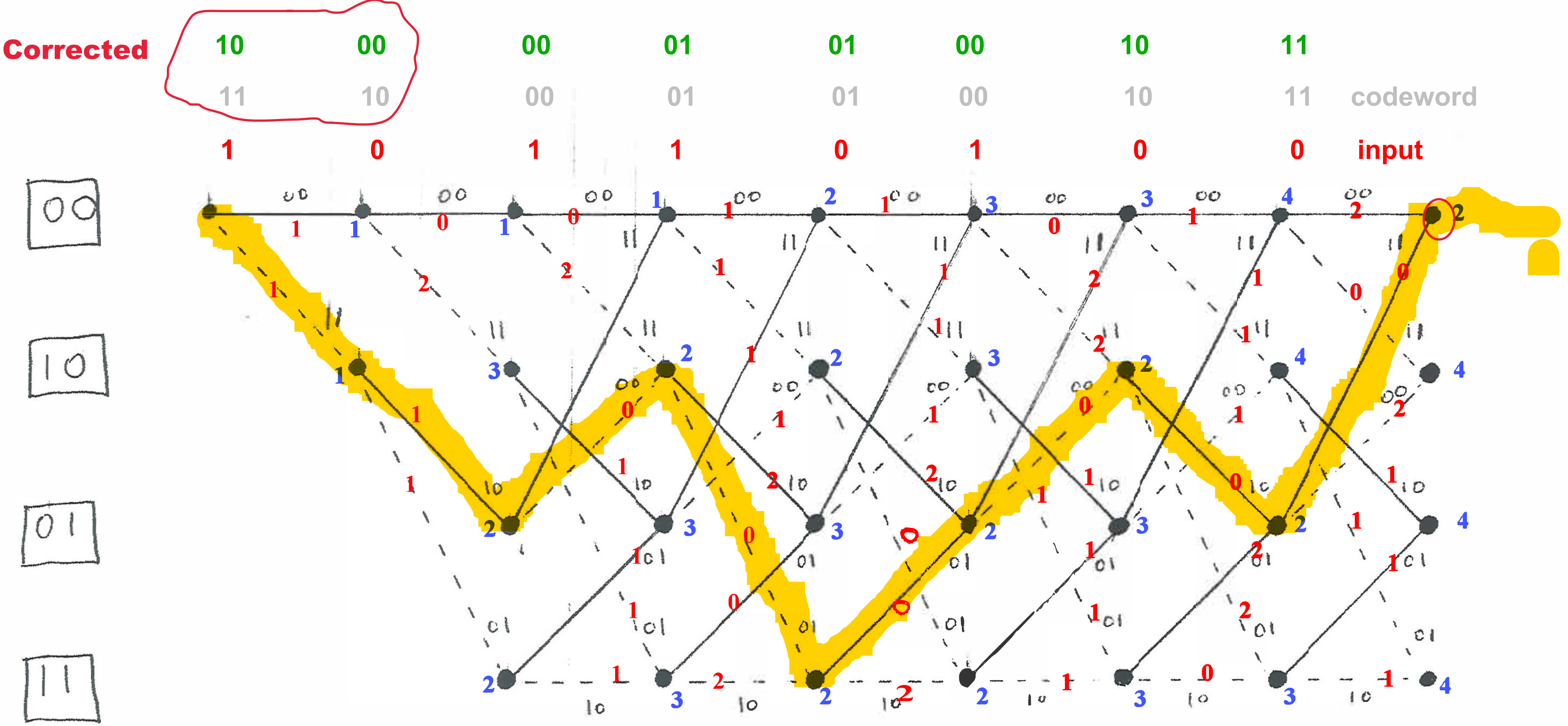
10	00	00	01	01	00	10	11	received
11	10	00	01	01	00	10	11	codeword
1	0	1	1	0	1	0	0	input




10 00 00 01 01 00 10 11 received
11 10 00 01 01 00 10 11 codeword
1 0 1 1 0 1 0 0 input
00
10
01
11


10 00 00 01 01 00 10 11
11 10 00 01 01 00 10 11 codeword
1 0 1 1 0 1 0 0 input
00
10
01
11


10 00 00 01 01 00 10 11
11 10 00 01 01 00 10 11 codeword
1 0 1 1 0 1 0 0 input
00
10
01
11


Corrected
10 00 00 01 01 00 10 11
11 10 00 01 01 00 10 11 codeword
1 0 1 1 0 1 0 0 input
00
10
01
11

The last 2 zeros are forced to go back to 00

* <u>What is the true rate of the convolutional code?</u>

$B=6,\ K=1$

- BK is the # of (pure) info bits. (In our example, 6, because)
- # of flush input bits is $\nu_{max} \cdot K$
- # of flush output bits is $\nu_{max} \cdot m$

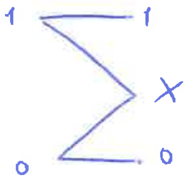
B = time slots over which we are sending bits of actual data.

The actual length of the codeword is

$$B \cdot m + \nu_{max} \cdot m = m(B + \nu_{max})$$

Rate: $\dfrac{KB}{m(B+\nu_{max})} \xrightarrow{B\to\infty} \dfrac{K}{m}$ in the example $R = \dfrac{1.6}{2(6+2)} = \dfrac{3}{8}$

Channel in the Internet: Not flip, but loss model



BSC $\rightarrow$ AWGN

AWGN decoding of convolutional codes

$$u(t) \longrightarrow v(t)$$

$t: 0 \to N-1$

$0 \to \sqrt{E_s}$, $\quad 1 \to -\sqrt{E_s}$

$s(t)$

$$\xrightarrow[n(t)]{\text{AWGN}} \quad r_J(t) = s_J(t) + n_J(t)$$

$\{n_J(t)\}$

$n_J(t) \sim$ AWGN $\sim N(0, \sigma^2) = N(0, \frac{N_0}{2})$

we consider it real.

$$\max_{t,J} P\left(\{r_J(t)\} \,\middle|\, \{s_J(t)\}\right) \qquad J = 1 \to m, \; t = 0 \to N-1$$

$\longrightarrow$ independence

$$\max_{s} \sum_{t=0}^{N-1} \sum_{J=1}^{m} \ln\left[P\left(r_i(t) \,\middle|\, s_i(t)\right)\right]$$

$P\left(r_i(t) \mid S_i(t)\right) = P_r$ (noise $n(t)$ is @ a value $r_i(t) - S_i(t)$ away from its zero mean)

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{-(x-\mu)^2}{2\sigma^2}} = \cancel{\frac{1}{\sqrt{2\pi \frac{N_0}{2}}}} e^{\frac{-(r_i(t)-S_i(t))^2}{\cancel{2 N_0/2}}}$$

$$\max \sum_{t=0}^{N-1} \sum_{J=1}^{m} -\left(r_i(t) - S_i(t)\right)^2 = \sum_{t=0}^{N-1} \sum_{J=1}^{m} \underbrace{-\cancel{r_i^2(t)}}_{\text{Constant}} + 2 r_i(t) S_i(t) - \underbrace{\cancel{S_i^2(t)}}_{E_S}$$

$$= \max \sum_{t=0}^{N-1} \sum_{J=1}^{m} r_i(t) S_i(t) = \max_{\underline{V}} \sum_{t=0}^{N-1} \sum_{J=1}^{m} r_i(t) (-1)^{V_i(t)}$$

we assumed BPSK at the beginning

Same process as the BSC except now metric is not d_H (Hamming distance) (at each t).

(again key to this is independence of noise t, i)

$$\sum_{i=1}^{m} r_i(t) (-1)^{V_i(t)} \longleftarrow \text{our new metric.}$$

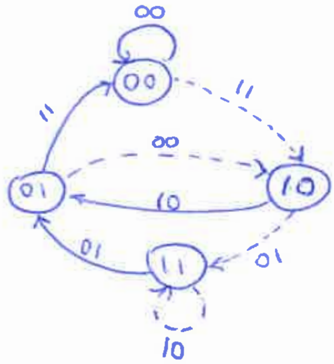
Decode with the Viterbi algorithm in the AWGN

$$v_1(t) = u(t) + u(t-1) + u(t-2)$$

$$v_2(t) = u(t) + u(t-2)$$

$$G(D) = [1 + D + D^2 \quad 1 + D^2]$$

This gave the state diagram



in		triplet	output
0 :	00 → 00	000	00
0 :	10 → 01	010	10
	01 → 00	001	11
	11 → 01	011	01
1 :	00 → 10	100	11
	10 → 11	110	01
	01 → 10	101	00
	11 → 11	111	10

Branch metric $\sum r_i(t)(-1)^{v_i(t)}$

$$v_i(t) \Rightarrow \begin{cases} 0 \rightarrow \sqrt{\mathcal{E}_s} \\ 1 \rightarrow -\sqrt{\mathcal{E}_s} \end{cases} = s_i(t)$$

$$r_i(t) = s_i(t) + n_i(t)$$

Example

to go back to 00

u: 0 1 1 0 1 1 0 0

v_1, v_2: 00 11 01 01 00 01 01 11 Assume $\mathcal{E}_s = 25$

(5,5) (-5,-5) (+5,-5) (+5,-5) (+5,+5) (+5,-5) (+5,-5) (-5,-5)

r: (3,4) (-2,0) (4,-2) (4,-3) (3,4) (3,-5) (6,-4) (-4,-5)

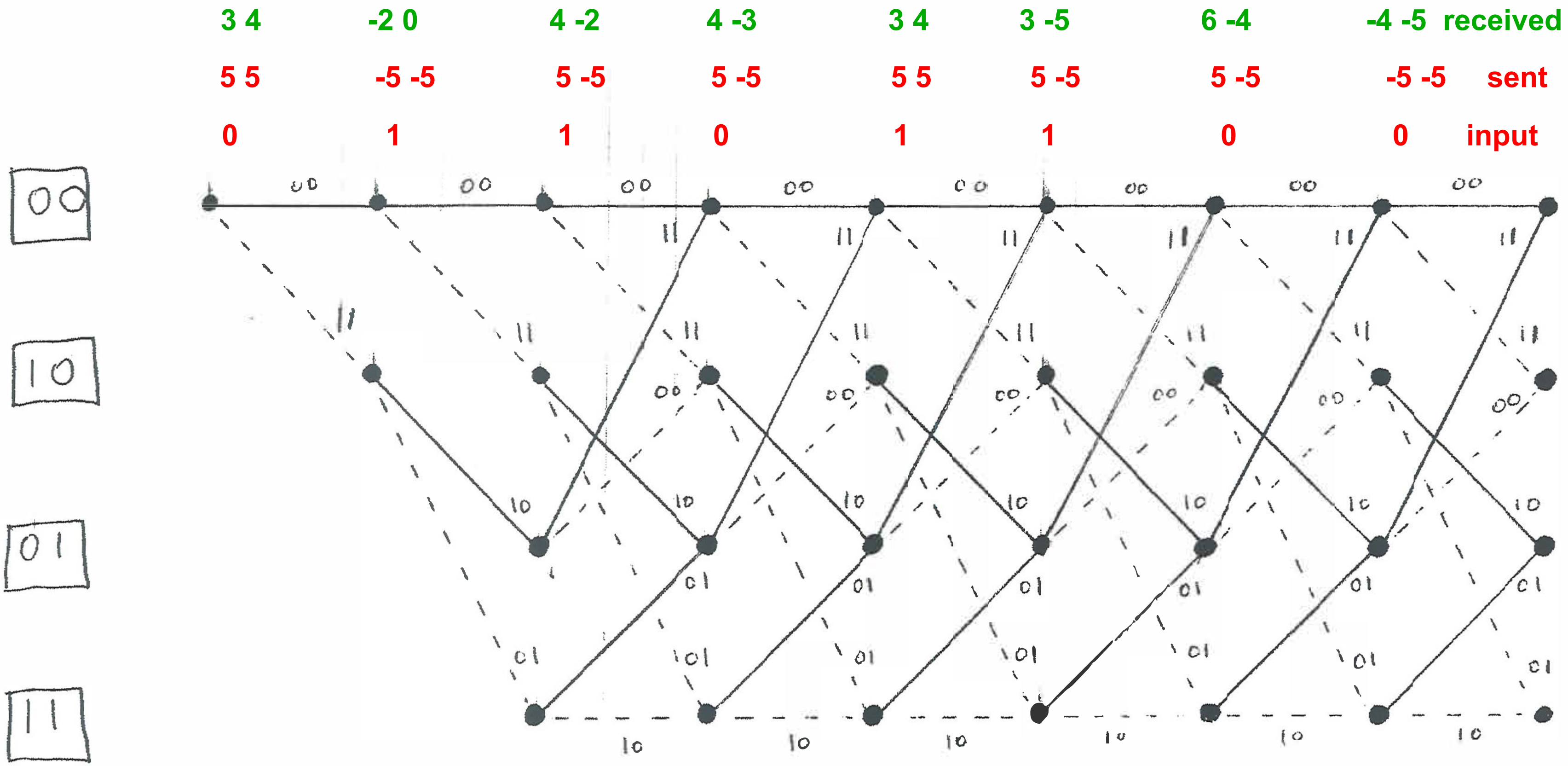
weights on the diagram: dot product

$(5\ \ 5)\begin{pmatrix}3\\4\end{pmatrix} \rightarrow (1\ \ 1)\begin{pmatrix}3\\4\end{pmatrix} \rightarrow$ $3(-1)^0 + 4(-1)^0 = 7 \rightarrow 1^{st}$ branch SUM

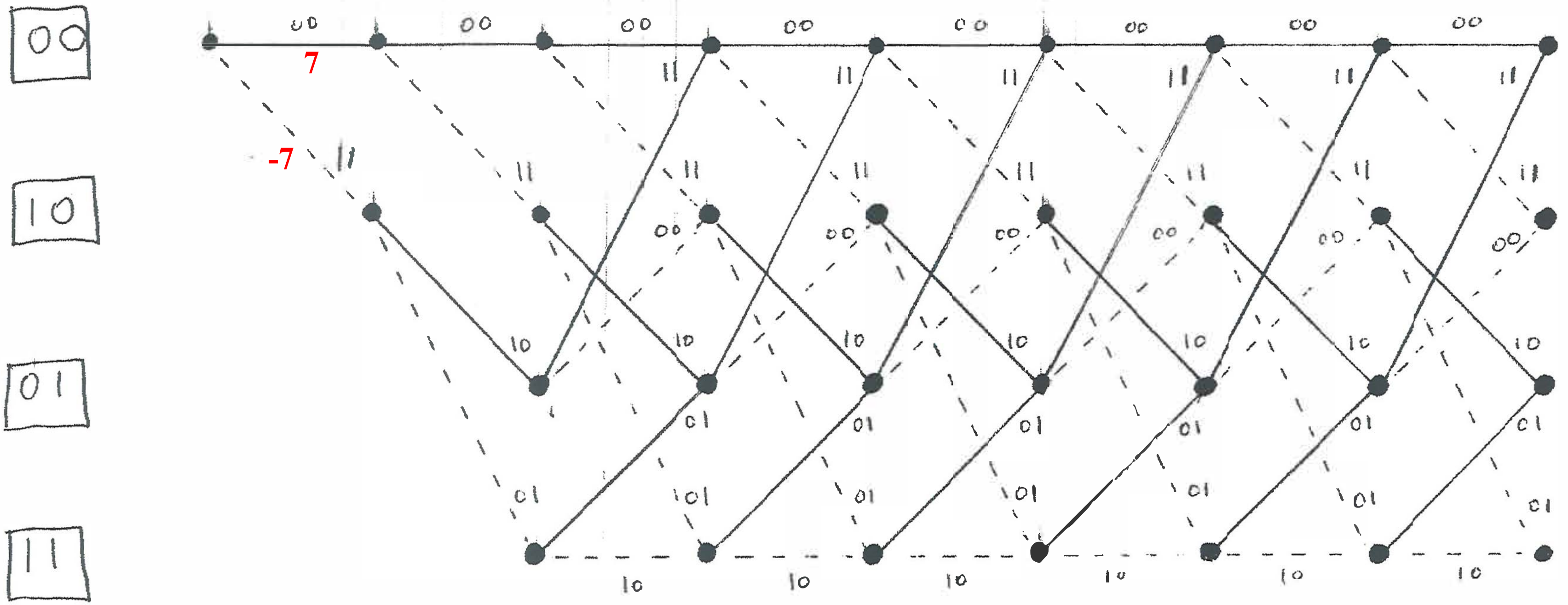
$3(-1)^1 + 4(-1)^1 = -7 \rightarrow 2^{nd}$ " FLIP & SUM

$3(-1)^0 + 4(-1)^1 = -1$ $1^{s} - 2^{nd}$

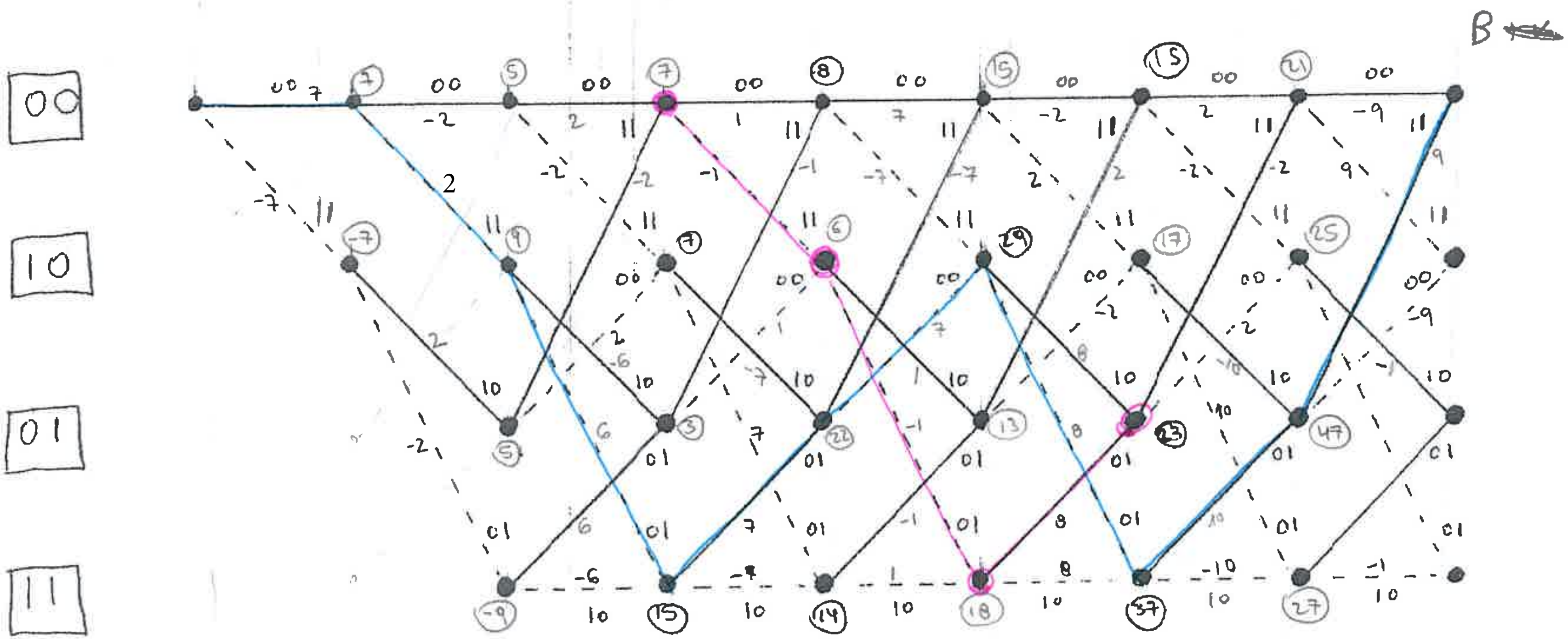
Decoded: 00 11 01 01 00 01 01 11


3 4 -2 0 4 -2 4 -3 3 4 3 -5 6 -4 -4 -5 received
5 5 -5 -5 5 -5 5 -5 5 5 5 -5 5 -5 -5 -5 sent
0 1 1 0 1 1 0 0 input
00
10
01
11

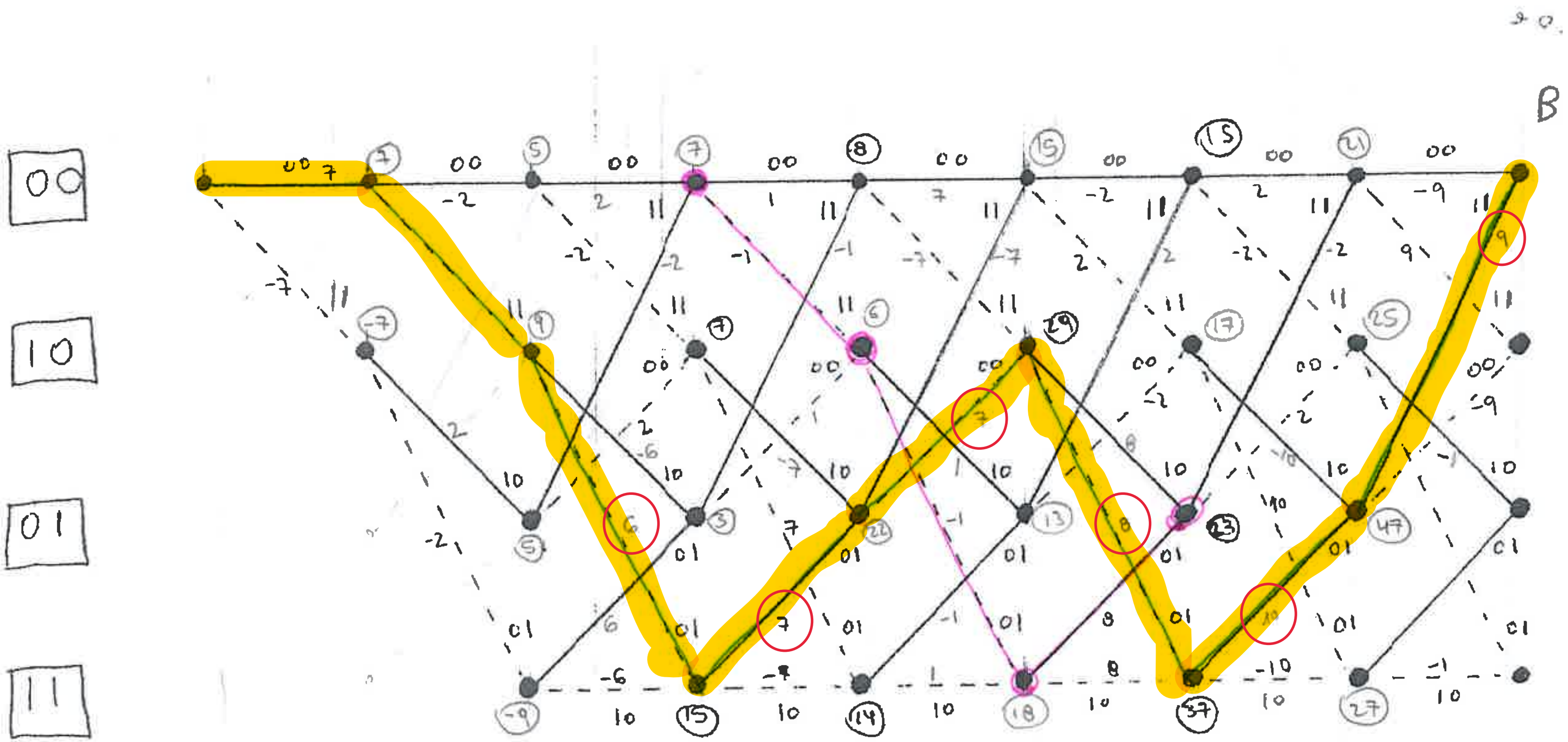
3 4	-2 0	4 -2	4 -3	3 4	3 -5	6 -4	-4 -5	received
5 5	-5 -5	5 -5	5 -5	5 5	5 -5	5 -5	-5 -5	sent
0	1	1	0	1	1	0	0	input



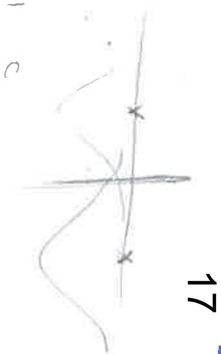
NOW AT NODE PICK MAXIMUM



u	01	1	1	0	1	1	0	0
v	00	11	01	01	00	01	01	11
Tx	(5, 5)	(-5,-5)	(5,-5)	(5,-5)	(5,5)	(5,-5)	(5,-5)	(-5,-5)
r	(3,4)	(-2,0)	(4,-2)	(4,-3)	(3,4)	(3,-5)	(6,-4)	(-4,-5)



u	01	1	1	0	1	1	0	0
v	00	11	01	01	00	01	01	11
Tx	(5, 5)	(-5,-5)	(5,-5)	(5,-5)	(5,5)	(5,-5)	(5,-5)	(-5,-5)
r	(3,4)	(-2,0)	(4,-2)	(4,-3)	(3,4)	(3,-5)	(6,-4)	(-4,-5)
Decoded	00	11	01	01	00	01	01	11



Nice things about these codes: The structure. We are going to study this a bit more now.

Weight distribution of convolutional code

$\{A_d\}_{d=0}^{m}$: A_d is # of codewords with Hamming weight d

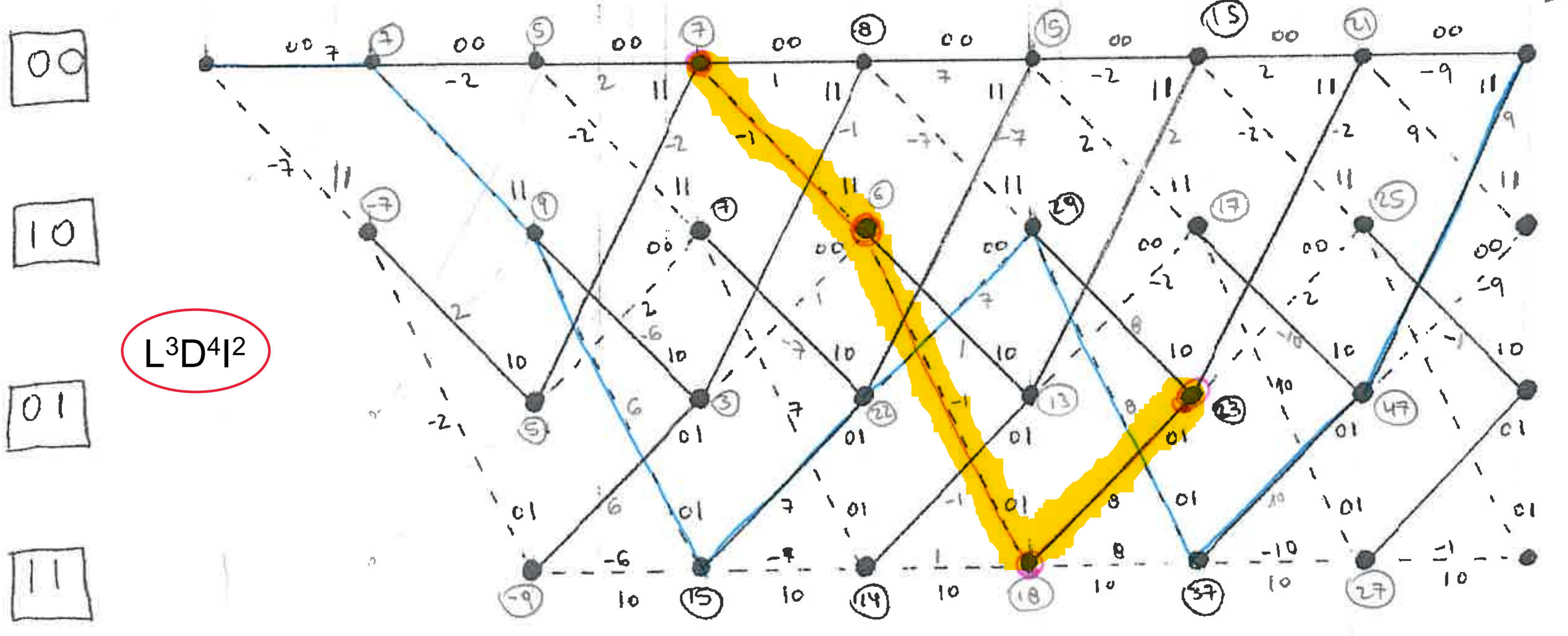
Can say more: Now we care about the paths:

$L^3 D^5 I^1 \longleftarrow$ Path $\begin{cases} \text{Length } 3 & \text{\# branches} \\ \text{Input: } 1 & \text{\# ---- lines} \end{cases}$
Output of weight 5 # 1 in the output

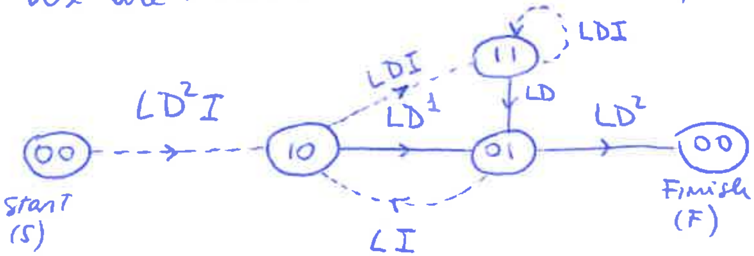
$$\sum_{i,j,k} a_{i,j,k} L^i D^j I^k \longrightarrow \text{\# of paths}$$

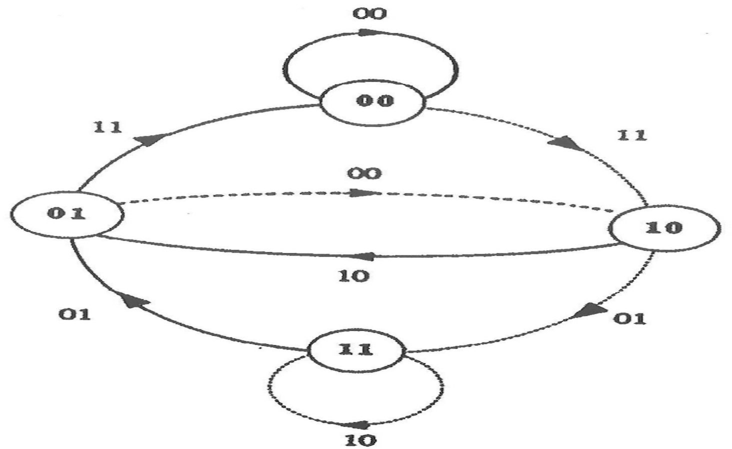
We will be looking to understand the # of paths that deviate from the ∀-zero state w/o visiting the ∀ zeros state in between.

HERE'S WHERE THE STRUCTURE OF THE CODE COMES INTO.


00
10
01
11
L3D4I2

We are interested in deviations from ∀ 0 path only





$$A_{10} = A_S \, LD^2I + A_{01} \, LI$$

$$A_{11} = A_{10} \, LDI + A_{11} \, LDI$$

$$A_{01} = A_{11} \, LD + A_{10} \, LD$$

Once

system of 3 eqs and 3 unknowns

$$A_S = 1$$

$$A_F = A_{01} \, LD^2$$

Solve for A_{01} and replace in

$$\underset{A}{\begin{pmatrix} A_{10} \\ A_{01} \\ A_{11} \end{pmatrix}} = \underset{M}{\begin{bmatrix} 0 & LI & 0 \\ LD & 0 & LD \\ LDI & 0 & LDI \end{bmatrix}} \underset{A}{\begin{pmatrix} A_{10} \\ A_{01} \\ A_{11} \end{pmatrix}} + \underset{B}{\begin{bmatrix} LD^2I \\ 0 \\ 0 \end{bmatrix}}$$

$$(I - M)A = B$$

$$\begin{bmatrix} 1 & -LI & 0 \\ -LD & 1 & -LD \\ -LDI & 0 & 1-LDI \end{bmatrix} \begin{bmatrix} A_{10} \\ A_{01} \\ A_{11} \end{bmatrix} = \begin{bmatrix} LD^2 I \\ 0 \\ 0 \end{bmatrix}$$

Cramer's Rule

$$A_{01} = \frac{\begin{vmatrix} 1 & LD^2 I & 0 \\ -LD & 0 & -LD \\ -LDI & 0 & 1-LDI \end{vmatrix}}{\begin{vmatrix} 1 & -LI & 0 \\ -LD & 1 & -LD \\ -LDI & 0 & 1-LDI \end{vmatrix}}$$

$$A_{01} = \frac{1 \cdot 0 - LD^2 I(-LD(1-\cancel{LDI}) \quad LD(\cancel{LDI}))}{LI((-LD)(1-\cancel{LDI}) - LD(\cancel{LDI})) + 1(LDI)}$$

$$A_{01} = \frac{L^2 D^3 I}{-L^2 DI - LDI + 1} \Rightarrow A_{01} = \frac{L^2 D^3 I}{1 - LDI(1+L)}$$

$$\Rightarrow A_F = LD^2 A_{01} = \frac{L^3 D^5 I}{1 - LDI(1+L)}$$

$$\frac{1}{1-f(t)} = 1 + f(t) + f^2(t) + \cdots$$

$$A_F = L^3 D^5 I \left[1 + LDI(1+L) + L^2 D^2 I^2 (1+L^2) + \cdots \right] = L^3 D^5 I + L^4 D^6 I^2 + L^5 D^7 I^3 + \cdots$$

Setting $L=1$, $I=1$; you are getting information about the # of paths of given output weight (without regard to input weight and/or length)

Example: $4 \cancel{L^3} \cancel{I^2} D^5 + 3 \cancel{L^6} \cancel{I^3} D^5 = 7 D^5$

$$A_F(L,D,I)\Big|_{L=I=1} = \frac{L^3 D^5 I}{1 - LDI(1+L)}\Bigg|_{L=I=1} = \frac{D^5}{1-2D}$$

$$= D^5 (1 + 2D + 4D^2 + 8D^3 + 16D^4 + \cdots$$

$$= D^5 + 2D^6 + 4D^7 + 8D^8 + 16D^9 + \cdots$$

This gives the output weights of paths which go from the all-zero state w/o visiting the all-zero state in between.

$$P_{be} = \frac{1}{k} \sum_{i=1}^{k} \underbrace{P_{be,i}}_{\text{Prob that the } i^{th} \text{ message bit is in error}} = \frac{\text{Expected \# of erroneous messages}}{k}$$

$\llcorner$ probability of bit error

(Assume standard $\frac{k}{n}$ rate code)

$$P_{cwe} \leq \sum_{d=1}^{m} A_d (PEP)_d \quad \text{(union bound)}$$

probability of codeword error.

upperbound because the regions are not disjoint → counting areas + than once

PEP_d is pairwise error probability that a potential codeword $\underline{c}$ of Hamming weight d is decoded given that $\underline{0}$ was sent

$$(PEP)_d \leq \sum_{e=\lceil \frac{d}{2} \rceil}^{d} \binom{d}{e} \underbrace{\varepsilon^e (1-\varepsilon)^{d-e}}_{\text{prob. of a specific pattern of } e \text{ flipped bits.}}$$

for this type of error to happen, at least $\frac{d}{2}$ bits must flip.

$$P_{be} \leq \sum_{d=0}^{m} \sum_{w=0}^{k} [A_{w,d}]\, w (PEP_d)$$

how many bit errors make for each codeword error.

$\{A_{w\_d}\}^k$ — codewords of weight d with input w

$A_{wd}\ I^w D^d$ is the # of codewords in the code corresponding to input weight w and output weight d.

$$\boxed{\text{Going from codeword error prob to bit error prob.}}$$

input weight = the weight of the message vector underlying the codeword.

$$A_F(L, D, I)\Big|_{L=1} = \sum_{d,w} A_{w,d}\, I^w D^d$$

$$P_{be} \leq \frac{1}{k} \frac{\partial}{\partial I}\left(A_F(L=1, D, I)\right)\Bigg|_{\substack{D = \overbrace{2\sqrt{\varepsilon(1-\varepsilon)}}^{\circledast} \\ \underbrace{I=1}_{I_W}}}$$

Roughly speaking optimization.

Now for AWGN (we skip proof) Sending $\pm\sqrt{\varepsilon_s}$

$$P_{be} \leq \left(\frac{1}{k} \frac{\partial}{\partial I} A_F(L=1, D, I)\right)\Bigg|_{\substack{I=1 \\ D = e^{-E_s/N_0}}} \cdot Q\left(\sqrt{\frac{2\, d_{free}\, E_s}{N_0}}\right) \cdot e^{\frac{E_s d_{free}}{N_0}}$$

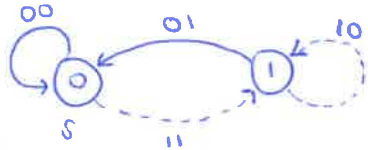
→ comes from error analysis in AWGN

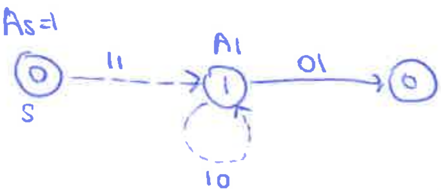
d_{free} is the minimum free distance of convolutional code of ∞ length
the # of smallest weight codewords.

Example

$$G(D) = [1 \ , \ 1+D]$$

$$v_1(t) = u(t)$$
$$v_2(t) = u(t) + u(t-1)$$





$$A_S = 1$$
$$A_1 = A_S\, LID^2 + A_1\, LID$$
$$= LID^2 + A_1 LID \Rightarrow$$
$$\Rightarrow A_1 = \frac{LID^2}{1-LID}$$
$$A_F = A_1 \cdot L \cdot D = \frac{L^2 I D^3}{1-LID}$$

We don't want a polynomial that is a ratio
We cannot interpret it

$$A_F = L^2 I D^3 (1 + LID + L^2 I^2 D^2 + \dots)$$

I want to calculate the free distance $\triangleq$ min weight of the codeword when you have ∞ length paths.

- I don't care about length $\to L=1$
- " " " " input $\to I=1$

$$A_F\Big|_{\substack{L=1\\I=1}} = \frac{D^3}{1-D} = D^3(1+D+D^2+D^3+\dots)$$

$$\therefore \boxed{d_{free} = 3}$$

Example

$R = \frac{1}{2} \quad k=1 \quad m=2$

$A_F(I, L, D) = L^4 I^2 D^5 + 2 L^6 I^4 D^7 + \dots$

Find upper bound on P_{be} in BSC when $\varepsilon \ll 1$

$$P_{be} \leq \frac{1}{k} \frac{\partial}{\partial I} A_F(L=1, D, I) \Big|_{D = 2\sqrt{\varepsilon(1-\varepsilon)},\, I=1}$$

we do care about the input weights but $I=1$ is the result of the optimization problem.

$$A_F\Big|_{L=1} = I^2 D^5 + 2 I^4 D^7 \quad \bullet \quad \frac{\partial}{\partial I}\left(A_F\Big|_{I=1}\right) = 2 I D^5 + 8 I^3 D^7 \Big|_{I=1}$$

$$\Rightarrow P_{be} \leq 2D^5 + 8D^7 \Big|_{D = 2\sqrt{\varepsilon(1-\varepsilon)}} = 2D^5(1 + 4D^2) = 2 \cdot 32\, \varepsilon^2 (1-\varepsilon)^2 \sqrt{\varepsilon(1-\varepsilon)}\,(1 + 16\varepsilon(1-\varepsilon))$$

$$= 64\, \varepsilon^2 (1-\varepsilon)^2 \sqrt{\varepsilon(1-\varepsilon)}\,(1 + 16\,\varepsilon(1-\varepsilon)) \underset{\varepsilon \ll 1}{\approx} 64\, \varepsilon^2 \sqrt{\varepsilon} = 64\, \varepsilon^{5/2}$$