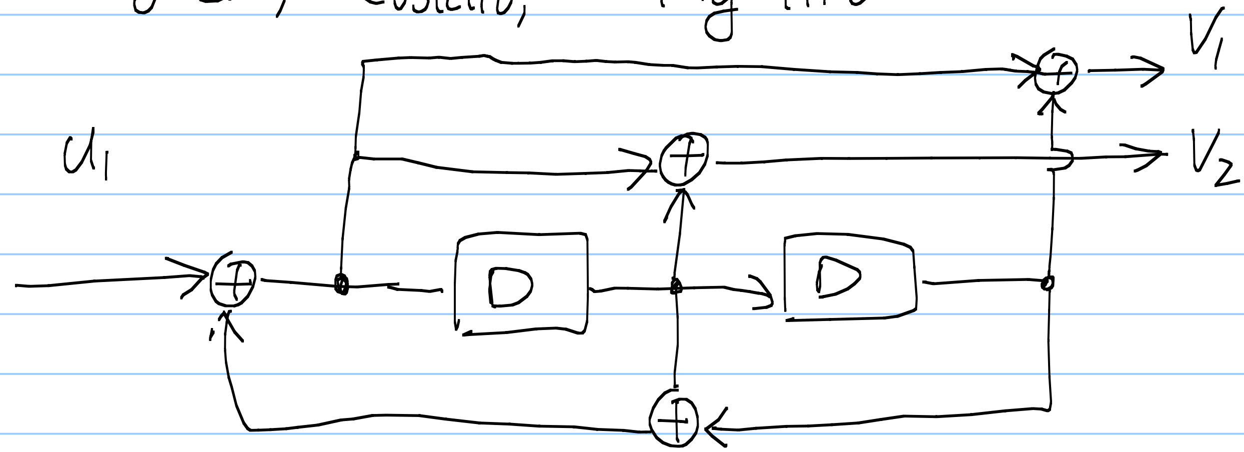
Lecture 08

Note Title 2/6/2012

A running example Lin, Costello, Fig 11.6



② The code rate is $\frac{\#\text{ of } u \text{ bits}}{\#\text{ of } v \text{ bits}}$

$= \frac{1}{2}$ in our example

③ Encoding complexity: $O(n(\text{Complexity of FSM}))$

④ Complexity of "optimal decoding"

$O(|\#\text{ states}| \times n)$

We will discuss it later.

③+④ ⇒ Scalable for large n, Long codewords have better performance.

⑤ Early pseudo-random number generators (PRNG) are based on shift registers with feedback.

⇒ the $\vec{X}$ codeword distribution looks random.

Recall random codes achieve the capacity. We expect good performance of the convolutional codes.

Unfortunately, the seq generated by shift registers is not random enough.

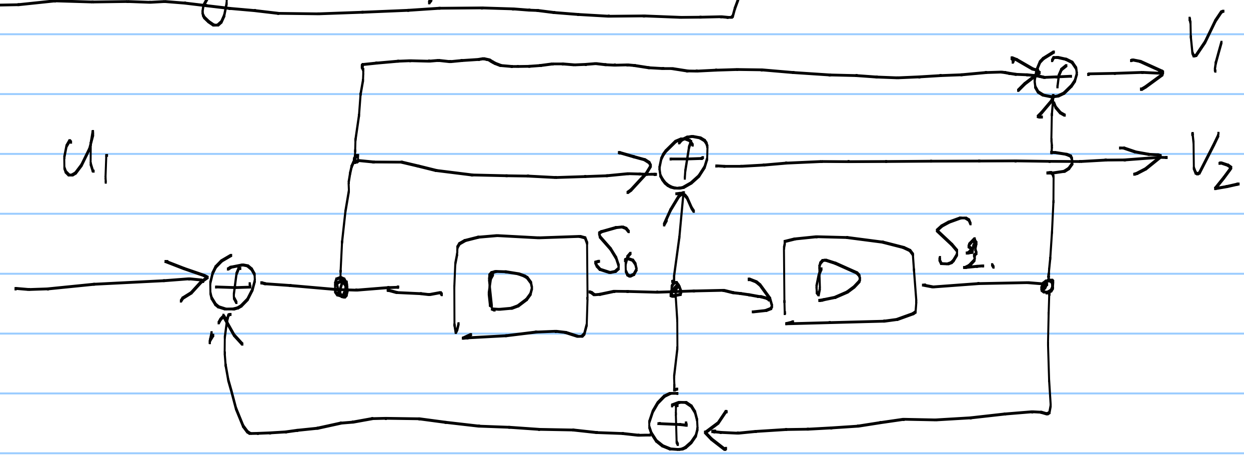
⇒ ① Not the best PRNG.

② The performance is still away from capacity.

* Optimal decoding (the Viterbi algorithm)

* A detour to the trellis representation

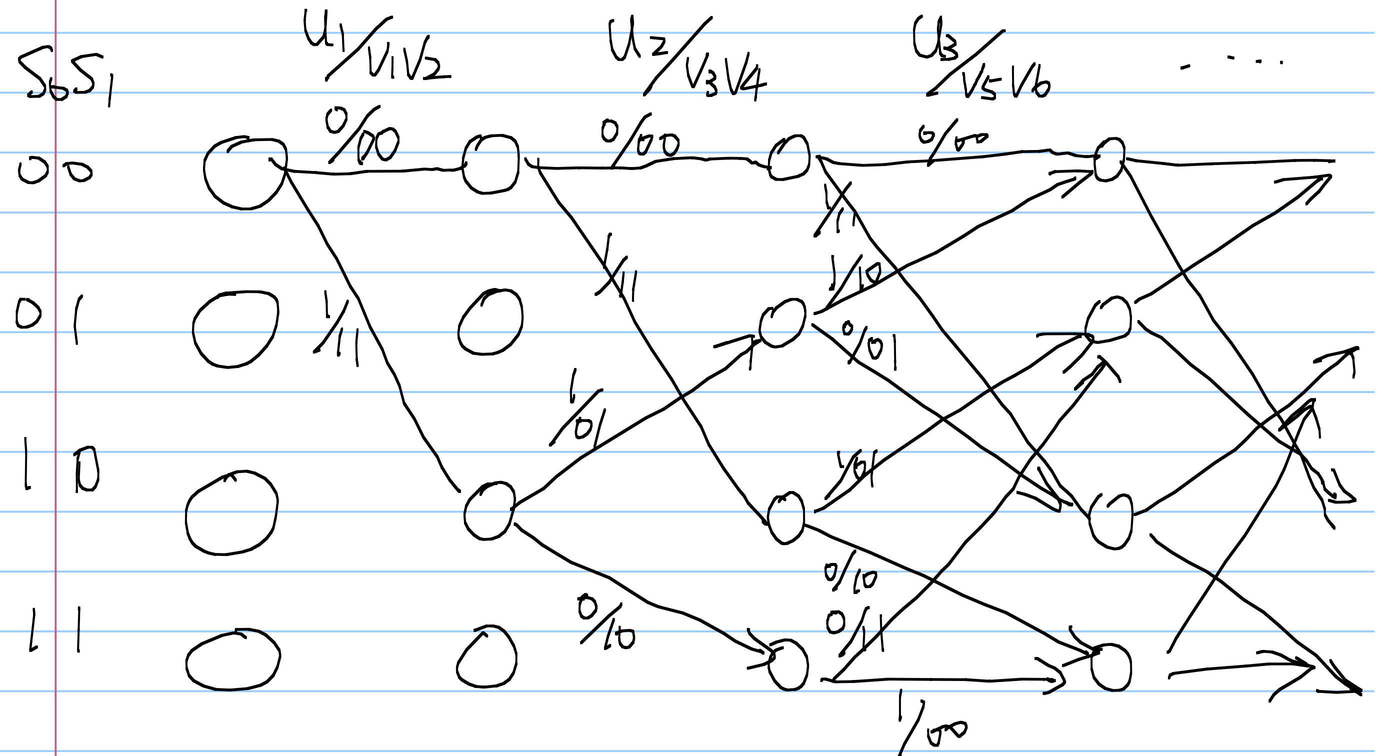
Shift register representation



$$\#\text{ of possible "states"} = 2^{\#\text{ delay units}}$$

The state is thus described by $S_0 S_1$, the bit values of the delay units.

Trellis representation



$u_1\ u_2\ u_3 \ldots$: the input message bits that "steer" the path

$v_1 v_2\ v_3 v_4\ v_5 v_6$ the positions of the output bit string that correspond to the selected segment of the path.

* The information bits steer the path along the trellis structure

* Each path corresponds to a codeword

* code rate : $\dfrac{\#\text{ of } u \text{ bits}}{\#\text{ of } v \text{ bits}}$

Our example is a rate $\frac{1}{2}$ convolutional code

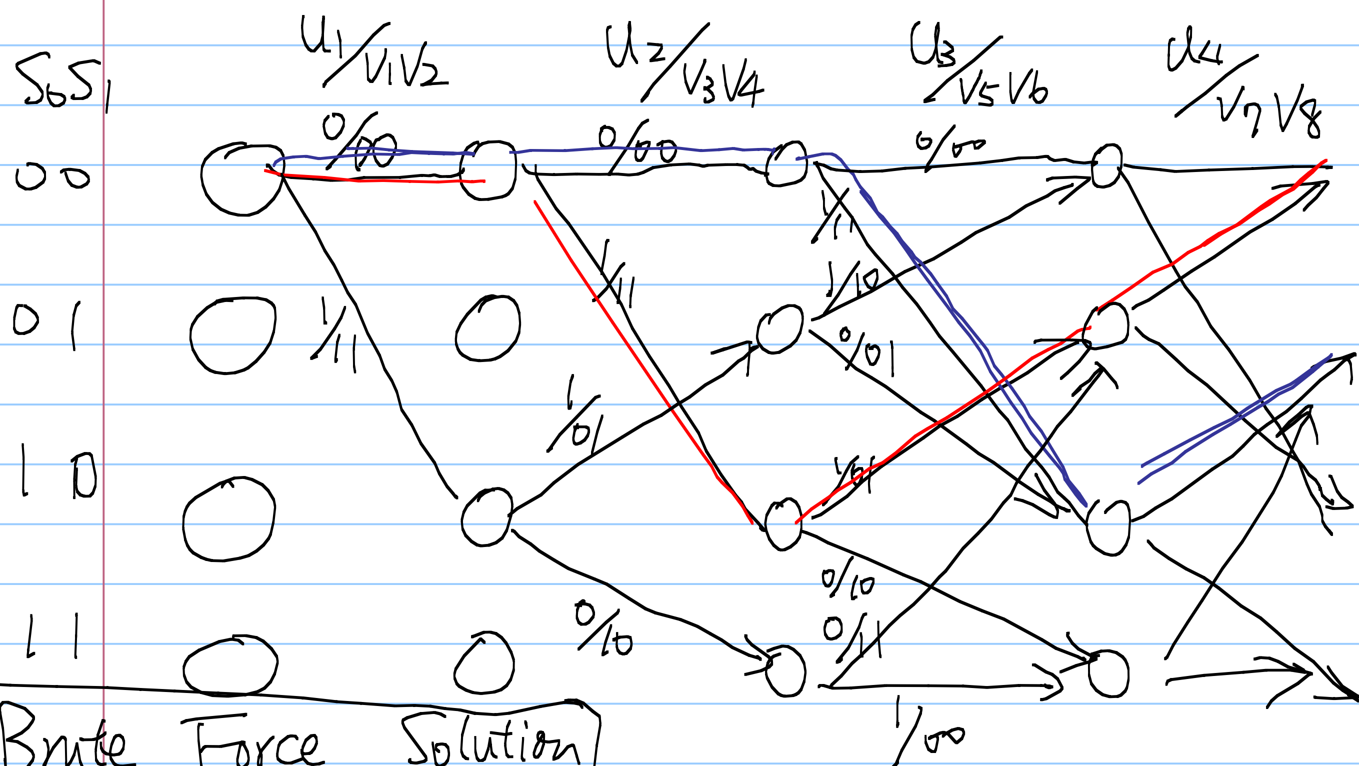
Optimal MAP decoder for the convolutional code

Example: we run the encoder for $n=4$ time slots.

$\Rightarrow$ info bits: $u_1\ u_2\ u_3\ u_4$

coded bits $v_1 v_2\ v_3 v_4\ \cdots\ v_7 v_8$

$\vec{x}_{0111} = 00\ 11\ 01\ 10$ $\quad$ $\vec{x}_{0011} = 00\ 00\ 11\ 01$



Brute Force Solution

Repeat the 16-Hypotheses testing

Find $\underset{\vec{x}_m}{\arg\max}\ P_{\vec{Y}|\vec{X}}(\vec{y}|\vec{x}_m)$ the ML decoder

The Viterbi Algorithm

1967

An efficient ML decoder for convolutional codes

Assume an i.i.d. channel.

Given any $\vec{y}_{obs}$, for any codeword $\vec{x}_m$ or for any given path p in the trellis, the likelihood value is

$$P_{y_1 y_2 | v_1 v_2} \cdot P_{y_3 y_4 | v_3 v_4} \cdot P_{y_5 y_6 | v_5 v_6} \cdot P_{y_7 y_8 | v_7 v_8}$$

if we let $\overline{p[1]}$ denote the output $v_1 v_2$ of the first segment. $\overline{p[i]}$ denote $v_{2i-1} v_{2i}$

$$= f_1(\overline{p[1]}) f_2(\overline{p[2]}) f_3(\overline{p[3]}) f_4(\overline{p[4]})$$

Where $f_i(\cdot)$ is a $\{0,1\}^2 \mapsto [0,1]$ function that is constructed by the observation $y_{2i-1} y_{2i}$

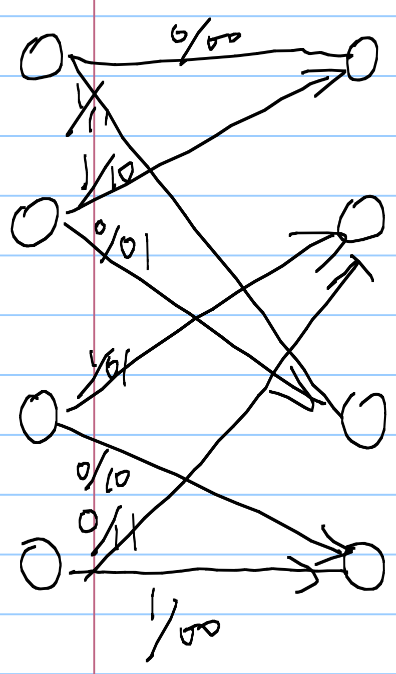
That is, given the observation $y_1, \cdots, y_8$ the functions $f_1(\cdot)$ to $f_4(\cdot)$ are decided by the likelihood $P(y_{2i-1}\, y_{2i} \mid \cdot, \cdot)$

The goal is to find a path

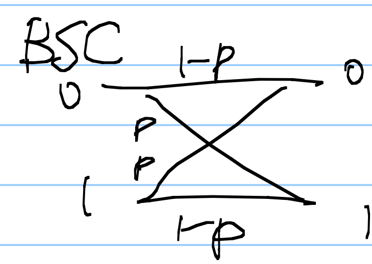
$p^* = (\overline{p[1]}, \overline{p[2]}, \overline{p[3]}, \overline{p[4]})$ that

maximizes

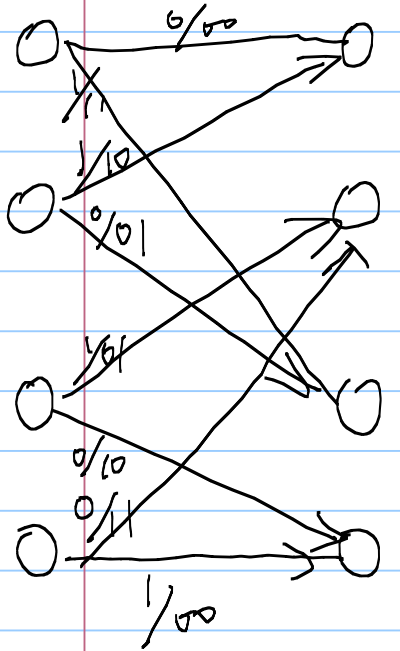
$$f_1(\overline{p[1]})\, f_2(\overline{p[2]})\, f_3(\overline{p[3]})\, f_4(\overline{p[4]})$$



ex: if BSC

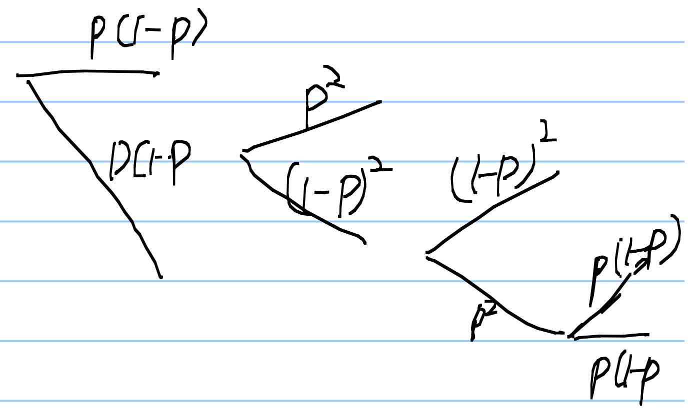


then



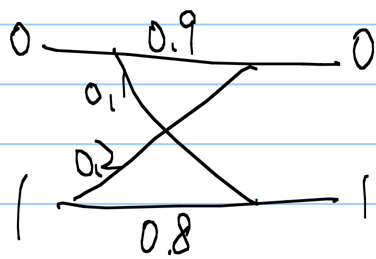
if $y_{2t-1}, y_{2t} = 0,1$

then



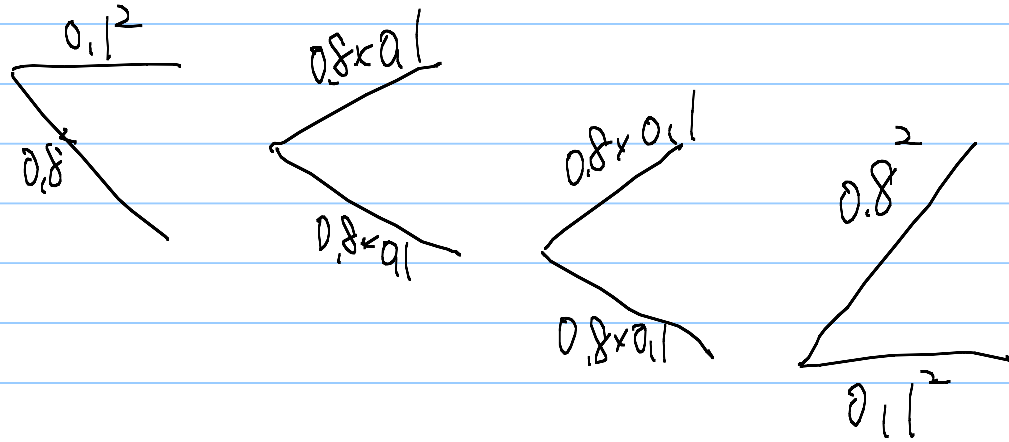
For different channel models & different observations, the partial objective function is different.

Ex:



$y_{2t-1}, y_{2t} = 1,1$

then



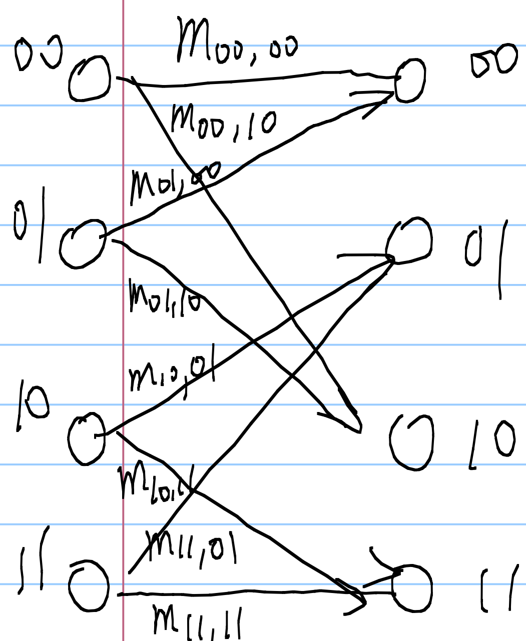
Ex: $P_{Y|X}(\cdot|x) \sim \mathrm{Gsn}((-1)^x, \sigma^2)$

$y_{2t-1}, y_{2t} = (0.7, -\sqrt{2})$

Then

$$\frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(0.7-1)^2}{2\sigma^2}} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(-\sqrt{2}-1)^2}{2\sigma^2}}$$

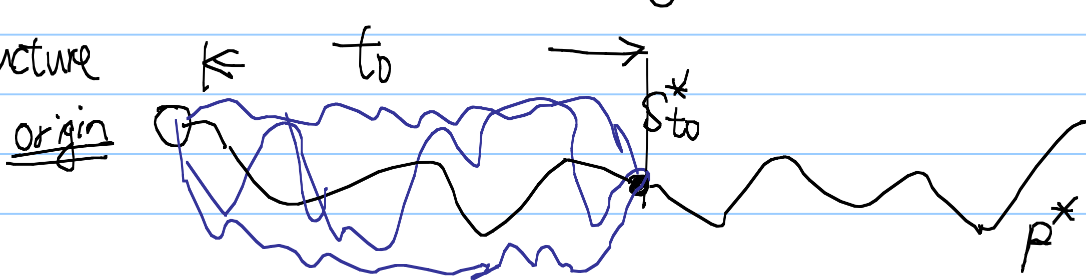
$$\frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(0.7-(-1))^2}{2\sigma^2}} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(-\sqrt{2}-(-1))^2}{2\sigma^2}}$$



In the end, each segment has a metric $m_{s_t, s_{t+1}}$

* The VA algorithm

Observation: If a path p^* maximizes $\prod_{t=1}^{n} f_t(p[t])$ among all paths. let $S^*_{t_0}$ denote the state that p^* is using at the t_0-th stage ~~of~~ the trellis structure



then the partial path $0 \xrightarrow{P^*} S^*_{t_0}$ is also the partial path maximizing

$$\prod_{t=1}^{t_0} f_t(\overline{P[t]})$$

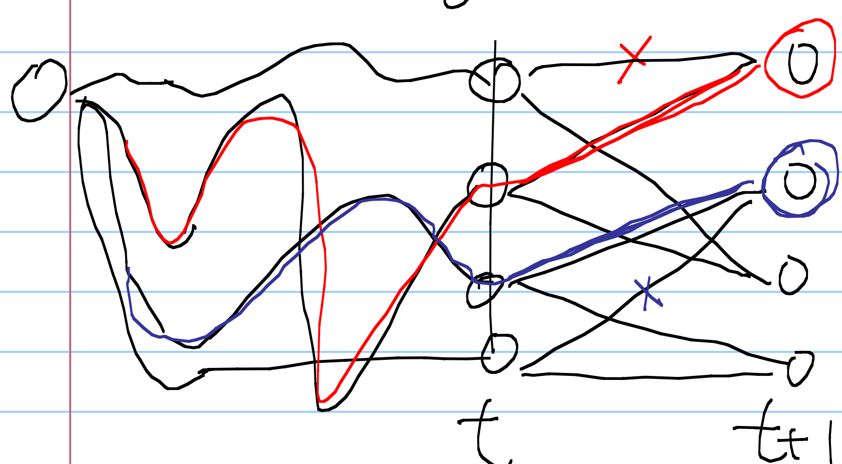
among all partial paths from 0 to $S^*_{t_0}$

(This observation is also the basis of the Dijkstra shortest path algorithm & the dynamic programming.)

* The Viterbi algorithm is then described as follows. (A high-level description)

① Intermediate goal: Keep track of the maximizing partial path from the origin to each state in the t-th stage.

② For the $(t+1)$th stage, update the new maximizing partial path for each state, based on the results from the t-th stage ⟶ Forward iteration



③ In the final stage, compare the maximizing complete paths for individual states & select the globally maximizing path.

Detailed implemenentation of the Viterbi algorithm

① Compute the partial objective function for the given observation y_{2t-1}, y_{2t} for each t

See many previous examples

& we denote the metrics for each segment as $m_{s,s'}$

② Compute the forward metric $a_s[t]$.

For the zero-th stage ($t=0$) set the metric $a_{00}[t]=1$ & $a_s[t]=0$ for all other states $S \neq 00$. In our running example, we have

$a_{00}[0]=1, \quad 0 = a_{01}[0] = a_{10}[0] = a_{11}[0]$

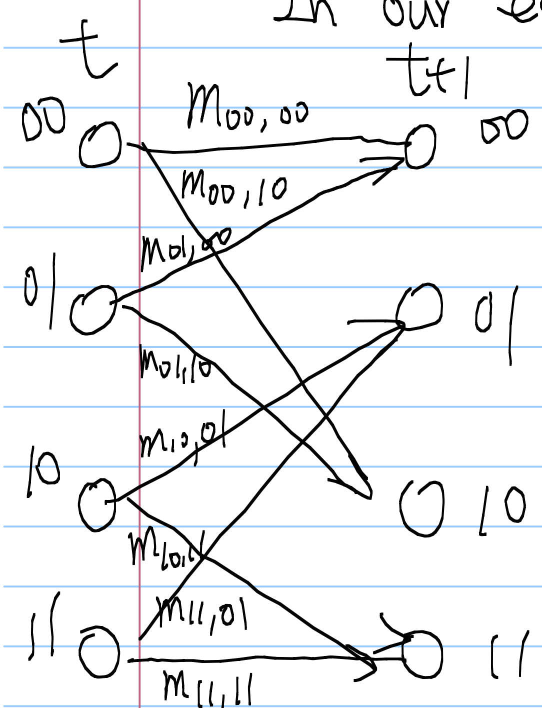
$a_s[t]$ is the metric of the maximizing partial path from the origin to S_t

$\because$ The origin stats from $S=00$

③ Update the $a_s[t+1]$ based on the following formula

$$a_s[t+1] = \max_{\substack{s' \text{ that} \\ \text{goes to } s}} \left\{ a_{s'}[t] \, m_{s',s} \right\}$$

In our example



$a_{00}[t+1] = \max\left(a_{00}[t]m_{00,00}, a_{01}[t]m_{01,00}\right)$

$a_{01}[t+1] = \max\left(a_{10}[t]m_{10,01}, a_{11}[t]m_{11,01}\right)$

$\vdots$

$a_{11}[t+1] = \max\left(a_{10}[t]m_{10,11}, a_{11}[t]m_{11,11}\right)$

④ After we update all $a_s[t]$, $t=0,\dots,T$.

the maximum value

$\max_{k} \prod_{t=1}^{T} f_t(\overline{p[t]}) = \max_{s} a_s[T]$, which is the maximum likelihood value of any codeword

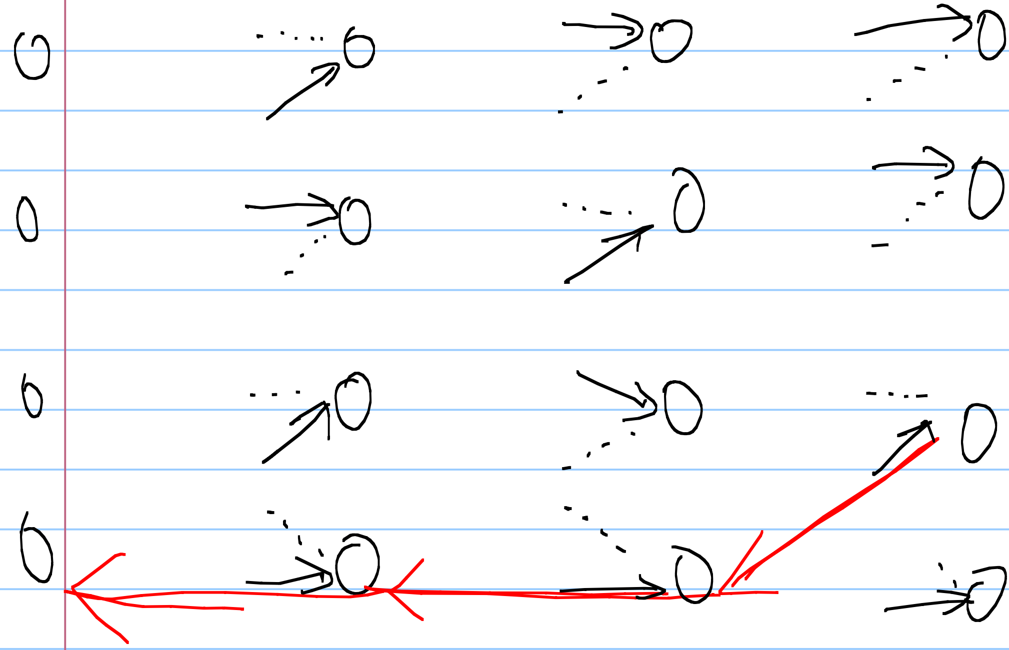
The above procedure describes how to find $\max_{k} \prod_{t=1}^{T} f_t(\overline{p[t]})$. But we are more interested in

$$\boxed{\operatorname{argmax} \prod_{t=1}^{T} f_t(\overline{p[t]}).}$$

which tells us the most likely codeword

Q: How to find the most likely codeword?

Ans: Similar to the Dijkstra algorithm. We memorize the choices of the max operations and then trace it backward from the destination back to the origin.



$a_{10}[I]$ is the largest