ELEC 6131: Error Detecting and Correcting Codes

Instructor:
Dr. M. R. Soleymani, Office: EV-5.125, Telephone: 848-2424 ext: 4103. Time and Place: Thursday, 17:45 – 20:15.
Office Hours: Thursday, 15:00 – 17:00

LECTURE 9: Viterbi Algorithm

Viterbi Algorithm

- While many algorithms have been proposed and used for decoding of convolutional codes, the Viterbi Algorithm (VA) is by far the most often used algorithm. Some of the reasons may be due to the very regular structure of this algorithm lending itself to modular implementation and fixed delay. The latter means that the VA decodes any sequence of a certain length with the same delay proportional to its length.

Viterbi Algorithm

- VA is a Maximum Likelihood (ML) algorithms, i.e., when presented with a received sequence say it tries to find the input sequence that has most likely generated.
- In short if it receives the vector $r = (r_0, r_1, \cdots, r_n)$ it outputs a codeword $v = (v_0, v_1, \cdots, v_n)$ such that:

$$P(r|v) \geq P(r|v') \text{ for any } v' \neq v.$$

Finally, the information sequence u that resulted in codeword v will be delivered to the user.

- Note that the VA does not maximize the probability of a codeword v being transmitted given that the received vector is r. It rather finds the codeword v that maximizes the chance of observing r.
- We discuss the difference between the two in the next slide.

Maximum *a posteriori* Probability (MAP) vs ML

- Assume that the vector v is transmitted with probability $P(v)$.
- $P(v)$ is called the *a priori* probability of v. That is, the probability of v before (prior to) observing anything about v.
- After receiving vector r at the receiver side, we can measure, the conditional probability $P(v|r)$. This is the probability that v has been sent given that r has been observed. This is called *a posterior (after the fact)* probability.
- MAP maximizes $P(v|r)$. Using, Bayes formula, we can write:

$$P(v|r) = \frac{P(v).P(r|v)}{P(r)}, \qquad \text{Eq. 1}$$

where $P(r) = \sum_v P(v)P(r|v)$.

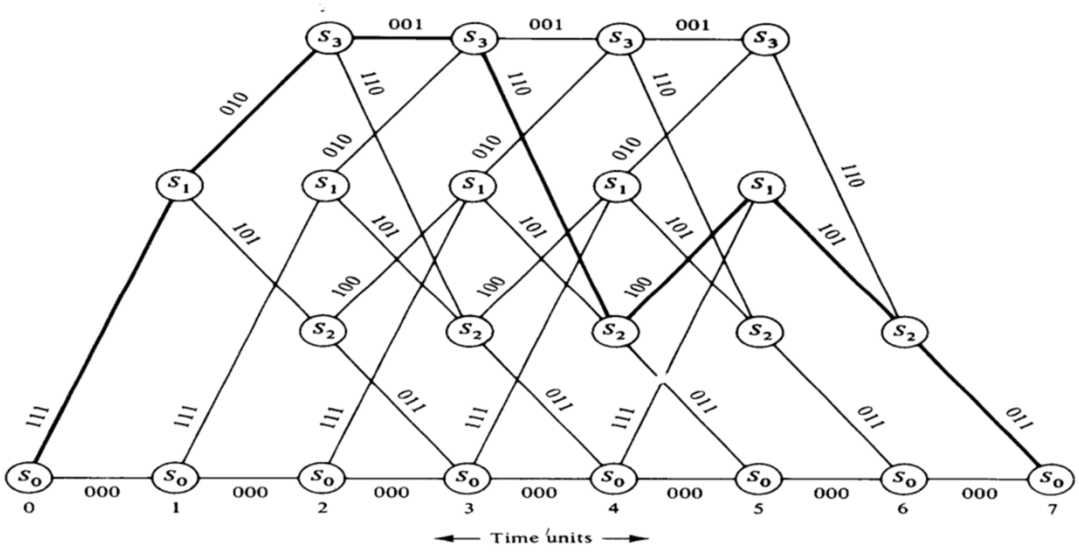
- In Equation 1, $P(r)$ is the same for all v and does not need to be considered. Also, if the probability of all vectors v at the input of the channel is the same, we can omit $P(v)$. As a result, we maximize $P(r|v)$. So, MAP become ML.

VA: Example

- Consider the rate ½ code (3, 1, 2) with:

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

- For the input sequence $u = (11101)$, the trellis is:



Likelihood Function

- Since the number of input bits is $h = 5$ and the memory size is $m = 2$, the trellis has 7 stages. The number of output bits is 21 (3×7) out of which 15 correspond to the input data and the rest are for clearing the memory before encoding the next packet.
- In general, for an (n, k, m) code, encoding h symbols, we have the input vector $\boldsymbol{u} = (u_0, u_1, \cdots, u_{h-1})$ of length $K = kh$ and the codeword $\boldsymbol{v} = (v_0, v_1, \cdots, v_{h+m-1})$ of length $N = n(h + m)$ and the received vector is: $\boldsymbol{r} = (r_0, r_1, \cdots, r_{h+m-1})$.
- The likelihood, i.e., the probability $P(\boldsymbol{r}|\boldsymbol{v})$ can be written as:

$$P(\boldsymbol{r}|\boldsymbol{v}) = \prod_{l=0}^{h+m-1} P(r_l|v_l) = \prod_{l=0}^{N-1} P(r_l|v_l)$$

Note that we have assumed that the channel is memoryless.

Log Likelihood Function

- Taking the logarithm of $P(r|v)$, we get:

$$logP(\boldsymbol{r}|\boldsymbol{v}) = \sum_{l=0}^{h+m-1} logP(r_l|v_l) = \sum_{l=0}^{N-1} logP(r_l|v_l)$$

- $logP(\boldsymbol{r}|\boldsymbol{v})$ is called the path metrics and denoted as $M(\boldsymbol{r}|\boldsymbol{v})$,

$$M(\boldsymbol{r}|\boldsymbol{v}) = \sum_{l=0}^{h+m-1} M(\boldsymbol{r_l}|\boldsymbol{v_l}) = \sum_{l=0}^{h+m-1} logP(\boldsymbol{r_l}|\boldsymbol{v_l})$$
$$= \sum_{l=0}^{N-1} M(\boldsymbol{r_l}|\boldsymbol{v_l}) = \sum_{l=0}^{N-1} logP(\boldsymbol{r_l}|\boldsymbol{v_l}).$$

$M(\boldsymbol{r_l}|\boldsymbol{v_l}) = logP(\boldsymbol{r_l}|\boldsymbol{v_l})$ are called the branch metrics, or bit metrics in the case of $\frac{1}{n}$ code.

Add, Compare, Select (ACS) Operation

- At any stage, we can compute a partial path metric for the first t branches of a path:

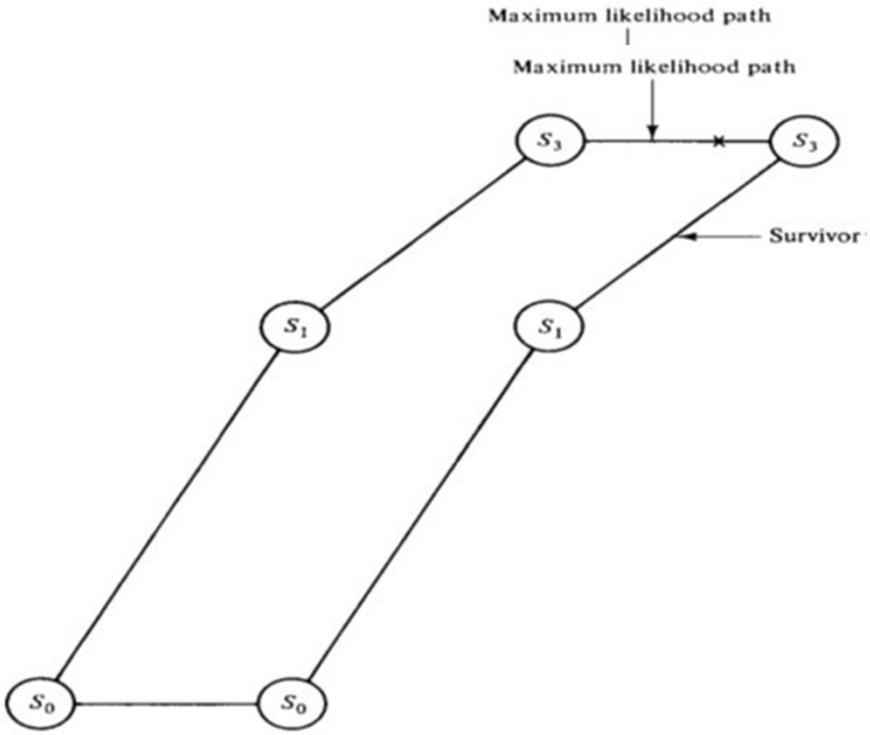
$$M_t(\boldsymbol{r}|\boldsymbol{v}) = \sum_{l=0}^{t-1} M(\boldsymbol{r_l}|\boldsymbol{v_l}) = \sum_{l=0}^{t-1} logP(\boldsymbol{r_l}|\boldsymbol{v_l})$$

$$= \sum_{l=0}^{nt-1} M(\boldsymbol{r_l}|\boldsymbol{v_l}) = \sum_{l=0}^{nt-1} logP(\boldsymbol{r_l}|\boldsymbol{v_l})$$

- Now to compute $M_{t+1}(\boldsymbol{r}|\boldsymbol{v})$ for a given path, we need to take the sum of the branch metrics of the first t stages for paths resulting in the present path and *add* to them the metric of the branch connecting each previous state to the present state and *compare* the resulting metrics and *select* the path with the maximum probability.

- A section a trellis is shown in the next slide in order to calify the above procedure.

Add, Compare, Select (ACS) Operation



- This is ***Add, Compare, Select*** algorithm summarized in the next slide.

Viterbi Algorithm

- The Viterbi Algorithm consists of the following steps:
- **Step 1:** Starting from $t = m$, Compute the partial metric for the single path coming to each state. Store the path (call it surviving path) and its metric.
- **Step 2:** Increase t by 1. Compute the partial metric for all 2^k paths entering each state by adding the branch metric entering that state to the metric of the connecting surviving path in the previous time unit. For each state compare the 2^k paths entering the state. Find the one with maximum probability (the survivor) and store it together with its metric. Eliminate all other paths.
- **Step 3:** If $t < h + m$ go to step 2; otherwise stop.
- A practical hint: Since the number of bits coded and decoded can be very large, in order to reduce the latency and to reduce the storage requirement after a certain number of stages, we can assume that the surviving paths for all state have converged for the few bit and release that bit and continue.

Viterbi Algorithm for BSC Channel

- Consider communication over a Binary Symmetric Channel with $p \leq \frac{1}{2}$ (the case of $p > \frac{1}{2}$ can easily be changed into $p \leq \frac{1}{2}$ by complementing the output bits).
- Assume that the distance between a received vector $\boldsymbol{r}$ and a codeword $\boldsymbol{v}$ is $d(\boldsymbol{r}, \boldsymbol{v})$. Then the likelihood is:

$$P(\boldsymbol{r}|\boldsymbol{v}) = p^{d(\boldsymbol{r},\boldsymbol{v})}(1-p)^{N-d((\boldsymbol{r},\boldsymbol{v})}$$

- The log likelihood function will be:

$$logP(\boldsymbol{r}|\boldsymbol{v}) = d(\boldsymbol{r}, \boldsymbol{v}) log \frac{p}{1-p} + Nlog(1-p).$$

- $Nlog(1-p)$ is a constant and we do not need to consider.
- Also: $log \frac{p}{1-p}$ is negative. So, maximizing the path metric $logP(\boldsymbol{r}|\boldsymbol{v})$ is equivalent to minimizing $d(\boldsymbol{r}, \boldsymbol{v})$:

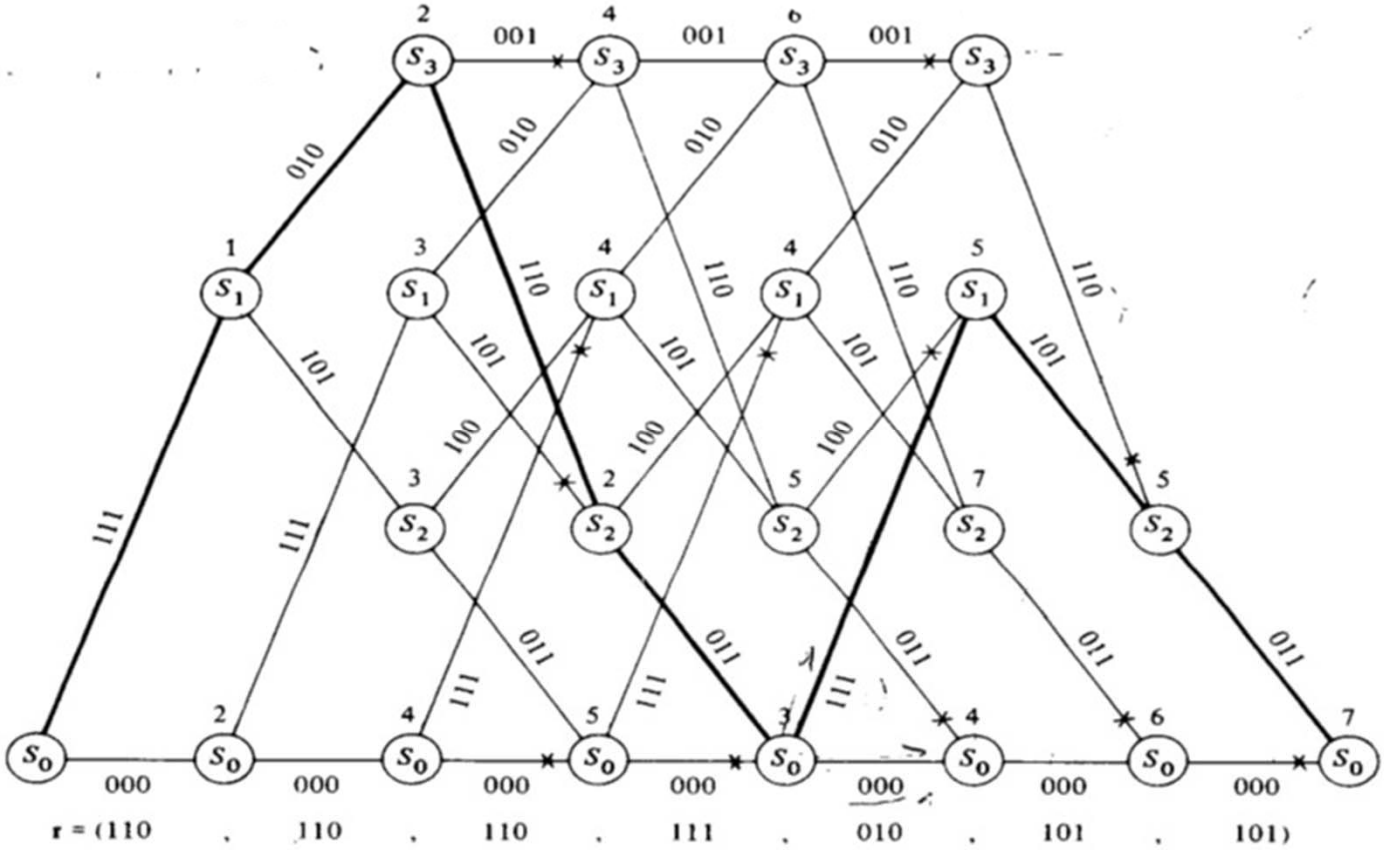
$$d(\boldsymbol{r}, \boldsymbol{v}) = \sum_{l=0}^{h+m-1} d(\boldsymbol{r_l}, \boldsymbol{v_l}) = \sum_{l=0}^{N-1} d(\boldsymbol{r_l}, \boldsymbol{v_l})$$

Viterbi Algorithm for BSC Channel

- The Viterbi Algorithm consists of the following steps:
- **Step 1:** Starting from $t = m$, Compute the Hamming distance from the single path coming to each state. Store the path (call it surviving path) and its distance.
- **Step 2:** Increase t by 1. Compute the partial Hamming distance for all 2^k paths entering each state by adding the branch metric (distance) entering that state to the metric of the connecting surviving path in the previous time unit. For each state compare the 2^k paths entering the state. Find the one with ***minimum distance*** (the survivor) and store it together with its distance. Eliminate all other paths.
- **Step 3:** If $t < h + m$ go to step 2; otherwise stop.

Viterbi Algorithm for BSC Channel: Example

- Consider the code with the following trellis:



Viterbi Algorithm for BSC Channel: Example

- Assume that the received vector is:

$$\boldsymbol{r} = (110, 110, 110, 111, 010, 101, 101)$$

- Following the Veterbi Algorithm, the codeword is

$$\hat{\boldsymbol{v}} = (111, 010, 110, 011, 111, 101, 011)$$

- It is shown as the highlighted path in the figure above.
- The decoded information sequence is $\hat{\boldsymbol{u}} = (11001)$.

Viterbi Algorithm for AWGN Channel

- Consider communications over Additive White Gaussian Noise (AWGN) channel. Assume that we use BPSK modulation, i.e., the transmitted signal is:

$$s(t) = \pm\sqrt{\frac{E_b}{T}}\cos(2\pi f_0 t)\,.$$

- This means that we have used the mapping:

$$1 \longrightarrow +\sqrt{E_b} \text{ and } 0 \longrightarrow -\sqrt{E_b}.$$

- Normalizing by $\sqrt{E_b}$, we consider the codeword:

$$\boldsymbol{v} = (v_0, v_2, \cdots, v_{N-1})$$

taking values ± 1 according to the mapping:

$$1 \longrightarrow +1 \text{ and } 0 \longrightarrow -1.$$

Viterbi Algorithm for AWGN Channel

- The received vector $\boldsymbol{r} = (r_0, r_1, \cdots, r_{N-1})$, where each component is a real-valued number consisting of the corresponding transmitted symbol plus an additive noise:

$$r_l = v_l + n_l.$$

- The noise being Gaussian, the conditional *probability density function (pdf)* of the normalized received symbol r_l given the transmitted bit v_l is:

$$p(r_l|v_l) = \sqrt{\frac{E_b}{\pi N_0}}\, e^{-\frac{\left(r_l\sqrt{E_b} - v_l\sqrt{E_b}\right)^2}{N_0}} = \sqrt{\frac{E_b}{\pi N_0}}\, e^{-\left(\frac{E_b}{N_0}\right)(r_l - v_l)^2}.$$

Where N_0 is the power spectral density of the noise.

Viterbi Algorithm for AWGN Channel

- Since the channel is white (memoryless), the log-likelihood of the received vector $\boldsymbol{r}$ given the transmitted codeword $\boldsymbol{v}$ is:

$$M(\boldsymbol{r}|\boldsymbol{v}) = \ln p(\boldsymbol{r}|\boldsymbol{v}) = \ln \prod_{l=0}^{N-1} p(r_l|v_l) = \sum_{l=0}^{N-1} \ln p(r_l|v_l)$$

$$= -\frac{E_b}{n_0} \sum_{l=0}^{N-1} (r_l - v_l)^2 + \frac{N}{2} \ln \frac{E_b}{\pi N_0}$$

- Note that $\frac{N}{2} \ln \frac{E_b}{\pi N_0}$ and $\frac{E_b}{n_0}$ are constants and do not have any effect on the maximization of $M(\boldsymbol{r}|\boldsymbol{v})$. Also, there is minus sign in front of the summation.
- So, in order to maximized $M(\boldsymbol{r}|\boldsymbol{v})$, we just need to minimize the Euclidean distance of $\boldsymbol{r}$ and $\boldsymbol{v}$:

$$d_E(\boldsymbol{r}, \boldsymbol{v}) = \sum_{l=0}^{N-1} (r_l - v_l)^2$$

Viterbi Algorithm for AWGN Channel

- The Viterbi Algorithm for AWGN channel consists of the following steps:
- **Step 1:** Starting from $t = m$, Compute the Euclidean distance from the single path coming to each state. Store the path (call it surviving path) and its distance.
- **Step 2:** Increase t by 1. Compute the partial Euclidean distance for all 2^k paths entering each state by adding the branch metric (distance) entering that state to the metric of the connecting surviving path in the previous time unit. For each state compare the 2^k paths entering the state. Find the one with ***minimum distance*** (the survivor) and store it together with its distance. Eliminate all the other paths.
- **Step 3:** If $t < h + m$ go to step 2; otherwise stop.

Viterbi Algorithm for AWGN Channel

- You may write the Euclidean Distance as:

$$d_E(\boldsymbol{r},\boldsymbol{v}) = \sum_{l=0}^{N-1}(r_l - v_l)^2 = \sum_{l=0}^{N-1} r_l^2 + \sum_{l=0}^{N-1} v_l^2 - 2\sum_{l=0}^{N-1} r_l v_l$$

$$= |\boldsymbol{r}|^2 + N - 2\sum_{l=0}^{N-1} r_l v_l = |\boldsymbol{r}|^2 + N - 2(\boldsymbol{r}.\boldsymbol{v})$$

- Since $|\boldsymbol{r}|^2 + N$ is independent of $\boldsymbol{v}$, instead of minimizing the Euclidean distance, we can maximize the correlation (the inner products) of the received vector $\boldsymbol{r}$ and the codeword $\boldsymbol{v}$:

$$\boldsymbol{r}.\boldsymbol{v} = \sum_{l=0}^{N-1} r_l v_l$$