The Viterbi Algorithm
EECS 869: Error Control Coding
Fall 2009

1 Background Material

1.1 Organization of the Trellis

The Viterbi algorithm (VA) processes the (noisy) output sequence from a state machine and determines the most likely input sequence. In other words, it "inverts"—or "undoes"—a state machine. Figure 1 shows an example of a state machine: the so-called (5,7) convolutional encoder (this will serve as our running example below). The input is a sequence of K information bits $\mathbf{m} = \{m_k\}_{k=0}^{K-1}$, where a *single bit* m_k is shifted in at each time step. The output is a sequence of $2K$ coded bits $\mathbf{c} = \{\mathbf{c}_k\}_{k=0}^{K-1}$, where a *pair of bits* $\mathbf{c}_k$ is shifted out at each time step (if we ever need to refer to individual bits within this pair, they are $c_k^{(1)}$ and $c_k^{(2)}$). The noisy channel output is a sequence of $2K$ values $\mathbf{r} = \{\mathbf{r}_k\}_{k=0}^{K-1}$, where a pair of values $\mathbf{r}_k$ is also produced at each time step (with individual values $r_k^{(1)}$ and $r_k^{(2)}$). Common noisy channel models are the binary symmetric channel (BSC) and the additive white Gaussian noise (AWGN) channel.

The (5,7) convolutional encoder in Figure 1 has two binary-valued memory elements (i.e., shift registers), which are labeled MSB (most-significant bit) and LSB (least-significant bit). The memory elements can assume four possible states: `00`, `01`, `10`, and `11`. In general, the state machine has a total of N_{S} possible states, where typically $N_{\text{S}} = 2^{\nu}$ with ν being an integer.

Every state machine has a corresponding *trellis diagram*. The trellis diagram for our example is shown in Figure 2. The trellis shows the $N_{\text{S}} = 4$ states and the allowable transitions between the states. Thus the trellis characterizes the manner in which the state machine behaves over time. The state transitions are referred to as *edges* or *branches* in the trellis diagram. In our example, the input to the state machine, m_k has 2 possible values, so there are 2 edges leaving each state (and two edges merging into each state). In general, if the input to the state machine is M-ary, then there are a total of $N_{\text{E}} = M \cdot N_{\text{S}}$ edges in the trellis diagram. The trellis is decorated with numerous labels, which we now define:

- *The starting state*: s^{S}. In general, the starting state takes on values $s^{\text{S}} \in \{0, 1, \cdots, N_{\text{S}} - 1\}$. It is convenient to label the states with an integer (decimal) value, rather than "vector" value such as `10`. This only requires us to come up with a one-to-one mapping (relabeling) between the two representations. In our example, the standard conversion between binary and decimal representations is the obvious choice, and this is used in Figure 2.

- *The ending state*: s^{E}. The ending state takes on values in the same range as the starting state, i.e., $s_{\text{E}} \in \{0, 1, \cdots, N_{\text{S}} - 1\}$. We use the same one-to-one mapping as before when going between the integer state representation and the vector state representation.

- *Edge input and output*: $m(e)$ and $\mathbf{c}(e)$: In Figure 2, each edge is labeled with values that are separated by a forward slash, i.e., $m(e)/\mathbf{c}(e)$. The value in front of the forward slash is the information bit $m(e)$ that causes the state machine to follow the given state transition. The value after the forward slash is the pair of output bits $\mathbf{c}(e)$ produced by the state machine as it follows the given state transition. The definition of the "output" will change from one type of state machine to the next—not all state machines output a pair of bits—so the labeling format will vary from case to case. For example, with trellis coded modulation (TCM) or continuous phase modulation (CPM), the state machine has different outputs.

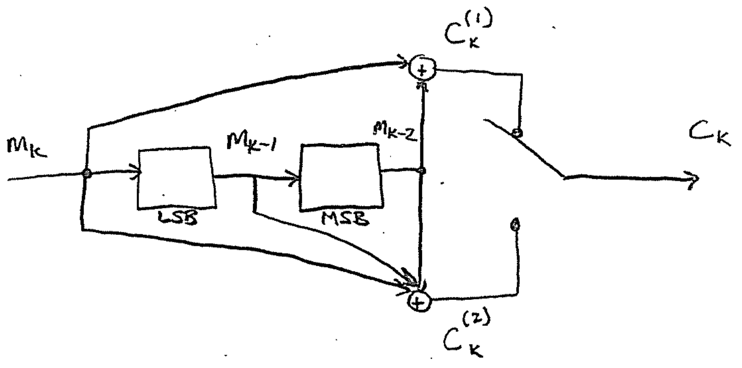

Figure 1: (5,7) convolutional encoder.

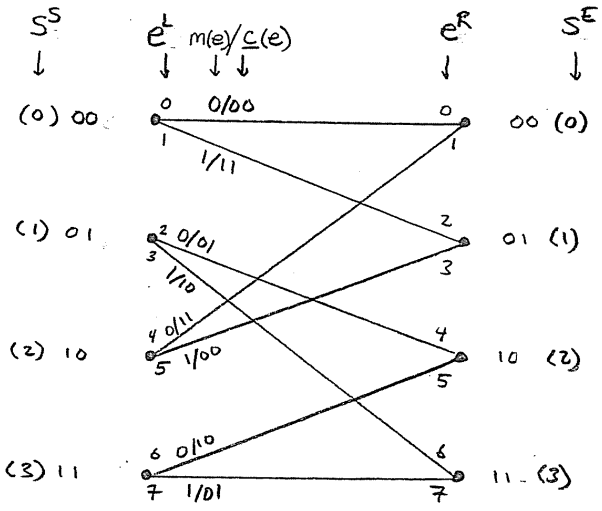

Figure 2: Trellis diagram for the (5,7) convolutional encoder.

- *The left edge index*: e^{L}. On the left-hand side of the trellis, each edge is labeled with a unique index $e^{\mathrm{L}} \in \{0, 1, \cdots, N_{\mathrm{E}} - 1\}$. Once again, we require some sort of one-to-one mapping between the integers and the edges. Because each edge is a unique combination of a starting state and an edge input, it make sense to combine these quantities into one value. The left edge indexes in Figure 2 are $e^{\mathrm{L}} = 2 \cdot s^{\mathrm{S}} + m(e)$. Stated in words, the state value is "left-shifted" (multiplied by 2) in order to "shift in" (add) the edge input in the LSB position.

- *The right edge index*: e^{R}. On the right-hand side of the trellis, each edge is labeled with another unique index $e^{\mathrm{R}} \in \{0, 1, \cdots, N_{\mathrm{E}} - 1\}$. For the left edge index, we made use of the *starting state* and the value that "appeared" at each state transition, i.e., the edge input. Now, for the right edge index, we will make use of the *ending state* and the value that "disappears" at each state transition. In our example, this is the MSB of the starting state. Thus, in Figure 2, the right edge indexes are $e^{\mathrm{R}} = 2 \cdot s^{\mathrm{E}} +$ the MSB of the edge's starting state.

Based on all of the above, the trellis diagram can be completely described by the information in Table 1. We point out that the rows in Table 1 are sorted according to increasing values of e^{L}. Thus, we have augmented the notation in Table 1 to reflect this ordering. For example, the starting state is $s^{\mathrm{S}}(e^{\mathrm{L}})$. An alternate version of Table 1 can be envisioned where the rows are sorted according to e^{R}.

Table 1: Trellis labels for the (5,7) convolutional encoder.

e^{L}	$s^{\mathrm{S}}(e^{\mathrm{L}})$	$m(e^{\mathrm{L}})$	$\mathbf{c}(e^{\mathrm{L}})$	$s^{\mathrm{E}}(e^{\mathrm{L}})$	$e^{\mathrm{R}}(e^{\mathrm{L}})$
0	0	0	00	0	0
1	0	1	11	1	2
2	1	0	01	2	4
3	1	1	10	3	6
4	2	0	11	0	1
5	2	1	00	1	3
6	3	0	10	2	5
7	3	1	01	3	7

1.2 Operation of the Viterbi Algorithm

With a richly described trellis, the operation of the VA can be described in a few succinct steps. Here are a few preliminary details and definitions:

- *State metric*: $A_k(s)$. A state metric is assigned to each state $s \in \{0, 1, \cdots, N_{\mathrm{S}} - 1\}$ at each time step $k \in \{0, 1, \cdots, K - 1\}$. Another name for the state metric is the *cumulative* metric. A major step within the VA is to update the state metric with a recursive operation (i.e., the new values of the state metric are a function of the previous values). Typically, it is not necessary to store a long history of past values of the state metric, the values at the current time step and the previous time step are usually sufficient, which minimizes memory requirements.

- *Metric increment*: $\gamma_k(e)$. The other part of the recursive state metric update is the metric increment. This is typically a function of the received data for the current time step (i.e., $\mathbf{r}_k$) and the state machine output for the given edge (i.e., $\mathbf{c}(e)$). The mathematical formulation of the metric increment will vary from one state machine to the next (e.g., convolutional codes, TCM, CPM) and also from one noisy channel model to the next (e.g., BSC, AWGN).

- *Traceback matrix*: $\mathbf{T}$. Unlike the state metric, a longer time history of the traceback matrix must be maintained. The individual elements of the traceback matrix are $T_k(s)$, which is indexed by both state and time, i.e., $s \in \{0, 1, \cdots, N_{\mathrm{S}} - 1\}$ and $k \in \{0, 1, \cdots, K - 1\}$. The individual traceback elements identify which of the merging paths at each ending state was declared the *survivor*. We will sometimes refer to $T_k(s)$ as the *local survivor* at state s and time step k, because each state has its own survivor. (At the end of the VA operation, on the other hand, we will be interested in determining the *global survivor*.) From a conceptual point of view, it is most simple to have a traceback matrix that spans the entire time interval $k \in \{0, 1, \cdots, K - 1\}$. However, this becomes infeasible when K becomes arbitrarily large. Thus, for practical purposes, it is often desirable to have a finite traceback length K_{T}, which is typically limited to a few tens of time steps.

Based on all of the above, the recursive state metric update is

$$A_k(s) = \min_{e: s^{\mathrm{E}}(e) = s} \left\{ A_{k-1}(s^{\mathrm{S}}(e)) + \gamma_k(e) \right\} \tag{1}$$

where $e : s^{\mathrm{E}}(e) = s$ is stated in words as "all edges such that the ending state of the edge is equal to s." This update is performed for all $k \in \{0, 1, \cdots, K - 1\}$ and for all $s \in \{0, 1, \cdots, N_{\mathrm{S}} - 1\}$. Conceptually, this can be thought of as two nested "for" loops: the outer loop goes over the time steps k and the inner loop goes over the states s. If the initial state of the encoder is known to be S_{-1}, then the initial values of the state metrics are

$$A_{-1}(s) = \begin{cases} 0, & s = S_{-1} \\ +\infty, & \text{otherwise.} \end{cases} \tag{2}$$

In other words, the known initial state is given an infinite "head start" over the rest of the states. If the initial state of the encoder is not known, then the state metrics are all initialized to zero.

The elements of the traceback matrix are given by

$$T_k(s) = \underset{e:s^{\mathrm{E}}(e)=s}{\arg\min} \left\{ A_{k-1}(s^{\mathrm{S}}(e)) + \gamma_k(e) \right\} \tag{3}$$

The traceback value is saved at the same point within the VA that the state metric is updated: the latter stores the minimum *value* of the merging metric while the former stores the *identity* of the edge with the minimum merging metric.

After $A_k(s)$ and $T_k(s)$ have been computed for all k and s, the final global survivor is identified as

$$\hat{S}_{K-1} = \arg\min_{s} \{A_{K-1}(s)\} \tag{4}$$

where the hat above $\hat{S}_{K-1}$ serves as a reminder that this is a quantity that is *estimated* at the receiver and may differ from the value at the transmitter S_{K-1} with some non-zero probability. In the case where S_{K-1} is known to the receiver (i.e., the state machine was fed with *termination symbols* to force it into a predetermined state), we use $\hat{S}_{K-1} = S_{K-1}$ instead of computing (4).

Using $\hat{S}_{K-1}$ as a starting point, the final step within the VA is to recursively *trace back* along the global surviving path via the operation

$$\hat{S}_{k-1} = s^{\mathrm{S}}(T_k(\hat{S}_k)) \tag{5}$$

for $k = K-1, \cdots, 1, 0$ and output whatever quantities are of interest at the receiver. Typically, this at least includes the estimated information sequence $\hat{\mathbf{m}} = \{m(T_k(\hat{S}_k))\}_{k=0}^{K-1}$.

1.3 Implementation Notes

Here are some final comments on the VA:

- A variety of different implementation architectures can be envisioned that differ in their use of the left edge index, right edge index, etc. These different implementations will ultimately be motivated by the targeted programming environment/platform. For example, MATLAB has a host of "vectorized" operations and functions that make it possible to implement the VA with a single (non-nested) "forward" loop to compute $A_k(s)$ and $T_k(s)$, followed by a single (non-nested) traceback loop. On the other hand, a C/C++ implementation will require the above-mentioned two nested loops, plus an additional inner loop to step through the edges required by the $\min\{\cdot\}$/$\arg\min\{\cdot\}$ operations in (1) and (3).

- We have formulated (1) and (3) with $\min\{\cdot\}$ and $\arg\min\{\cdot\}$. This is typically the case where the metric increments are used to compute some sort of distance metric and the *best* metric corresponds to the *minimum* distance. However, if the metric increments are used to compute, say, *correlation*, where the *best* metric corresponds to the *maximum* correlation, then $\max\{\cdot\}$/$\arg\max\{\cdot\}$ should be used in place of $\min\{\cdot\}$/$\arg\min\{\cdot\}$; also the state initialization in (2) should be negated.

2 Project

Complete the following tasks. You should submit an e-mail with three .m file attachments (use the e-mail address esp@eecs.ku.edu).

1. **Implement a Function to Organize the Trellis for Convolutional Codes.** You may assume that we are limited to rate $1/n$ feedforward non-systematic codes. You should implement this as a MATLAB function (without consulting any prior implementations by myself) with the following syntax:

   ```
   [sS, me, ce, sE, eR] = CreateCcTrellisXXX(G);
   ```

   where $\mathbf{G}$ is a $(\nu+1) \times n$ matrix that specifies n encoder functions with a constraint length (memory order) of ν. In our running example, we have a transfer function matrix

   $$\mathbf{G}(D) = \begin{bmatrix} 1+D^2 & 1+D+D^2 \end{bmatrix}$$

   and so we specify $\mathbf{G}$ in MATLAB as
   $$\mathbf{G} = \begin{bmatrix} 1 & 1 \\ 0 & 1 \\ 1 & 1 \end{bmatrix}.$$

   As for the output arguments, each is an $N_{\mathrm{E}} \times 1$ MATLAB vector (you might implement `ce` as a $N_{\mathrm{E}} \times n$ matrix). The elements within these output arguments are assumed to be in order of increasing left edge index. Thus, the specification for e^{L} is implied and does not need to output explicitly.

2. **Implement the Hard-Decision Viterbi Algorithm for Convolutional Codes.** You should implement this as a MATLAB function (without consulting any prior implementations by myself) with the following syntax:

   ```
   m_hat = VaCcHdXXX(r, sS, me, ce, sE, eR);
   ```

   where $\mathbf{r}$ is a $1 \times nK$ MATLAB vector containing hard decisions from the BSC (0's and 1's). The metric increment for the hard-decision VA is

   $$\gamma_k(e) = d_{\mathrm{H}}\big(\mathbf{r}_k, \mathbf{c}(e)\big)$$

   where $d_{\mathrm{H}}(\cdot,\cdot)$ denotes Hamming distance. In our running example, the metric increment is 0, 1, or 2. The objective of the VA is to *minimize* this metric.

3. **Implement the Soft-Decision Viterbi Algorithm for Convolutional Codes.** You should implement this as a MATLAB function (without consulting any prior implementations by myself) with the following syntax:

   ```
   m_hat = VaCcSdXXX(r, sS, me, ce, sE, eR);
   ```

   where $\mathbf{r}$ is a $1 \times nK$ MATLAB vector containing soft decisions from the AWGN channel (noisy $+1$'s and -1's). The metric increment for the soft-decision VA is

   $$\gamma_k(e) = \sum_{i=0}^{n-1} r_k^{(i)} a^{(i)}(e) = \langle \mathbf{r}_k, \mathbf{a}(e) \rangle$$

   where $\mathbf{a}(e)$ is the antipodal version of $\mathbf{c}(e)$ and $\langle \cdot,\cdot \rangle$ is the inner (dot) product between two vectors. Over time, the inner product increments amount to computing the correlation between $\mathbf{r}$ and $\mathbf{a}$. The objective of the VA is to *maximize* the correlation, so the appropriate changes must be made to the VA operations.