Example: Viterbi algorithm

This handout illustrates a specific example of the Viterbi algorithm with the purpose of unifying the concepts introduced in the application report, “Viterbi Decoding Techniques in the TMS320C54x Family”.

The convolution coder is often used in digital transmission systems where the signal to noise ratio is very low rendering the received signal error prone. The convolution coder achieves error free transmission by adding enough redundancy to the source symbols. The choice of the convolutional code is application dependent and varies with the frequency characteristics of the transmission medium as well as the desired transmission rate.

The Viterbi algorithm is a method commonly used for decoding bit streams encoded by convolution coders. The details of a particular decoder algorithm depends on the encoder. The Viterbi algorithm is not a single algorithm that can be used to decode coded bit streams of any convolution coder.

The system chosen to illustrate the encoding and Viterbi decoding process is a 4-state optimal rate 1/2 convolutional code. A block diagram of this convolution encoder is show in Figure 1. $x(n)$ is the input and $G_0(n)$ and $G_1(n)$ are the encoded outputs. The rate of the convolution coder is defined as the number of input bits to output bits. This system has 1 input and 2 outputs thereby resulting in a coding rate of 1/2. The number of states of a convolution coder is determined by its number of delay units. There are 2 delay units in this system and therefore, there are $2^2 = 4$ states. For a system with k delay units, there are 2^k states.

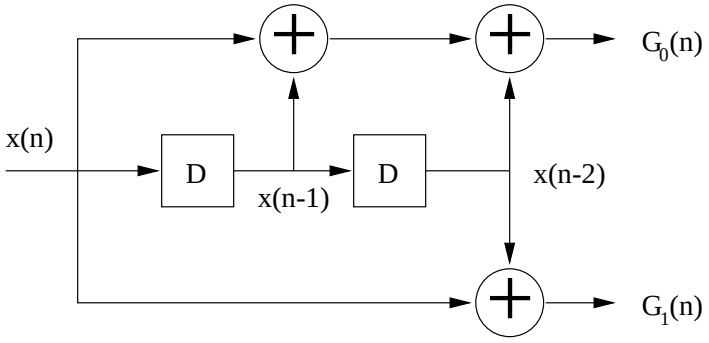

FIGURE 1. Block diagram of convolutional encoder

The system block diagram can be expressed with the following equations.

$$G_0(n) = x(n) + x(n-1) + x(n-2)$$

$$G_1(n) = x(n) + x(n-2)$$

The state diagram of this system is depicted in Figure 2. The states are defined as x(n-1),x(n-2) pairs and the state transitions are defined as $G_0(n), G_1(n)/x(n)$. The states are indicative of system memory. The state transitions gives you the path and the outputs associated with a given input. Since a state transition is associated with each possible input, the number of state transitions depends on the number of possible inputs. If there are m inputs, then there will be 2^m state transitions. The total number of state transitions at a point in time is the product of the number of

state transitions and the number of states, 2^{m+k}.

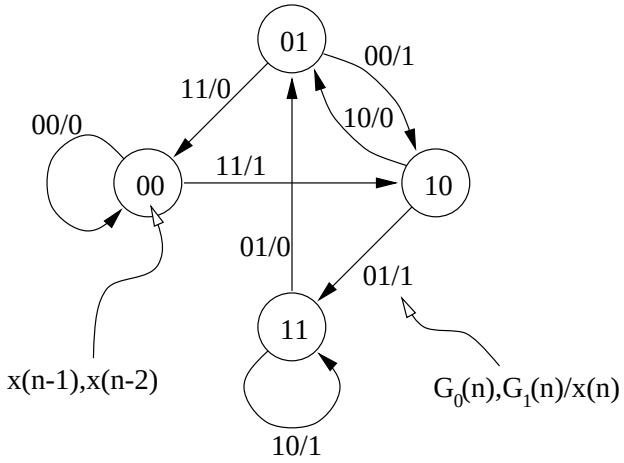

FIGURE 2. State diagram.

The state diagram offers a complete description of the system. However, it shows only the instantaneous transitions. It does not illustrate how the states change in time. To include time in state transitions, a trellis diagram is used (Figure 3). Each node in the trellis diagram denotes a state at a point in time. The branches connecting the nodes denote the state transitions. Notice that the inputs are not labeled on the state transitions. They are implied since the input, $x(n)$, is included in the destination states, $x(n)$, $x(n-1)$. Another thing to note is that the state transitions are fixed by the definition of the source, $x(n-1)$, $x(n-2)$, and the destination states. The state transitions are independent of the system equations. This a consequence of including the input, $x(n)$, in the state definition. In certain systems, definition of the states may be more complicated and may require special considerations. Explanation of such systems (such as the radix 4 trellis) is outside the scope of this example.

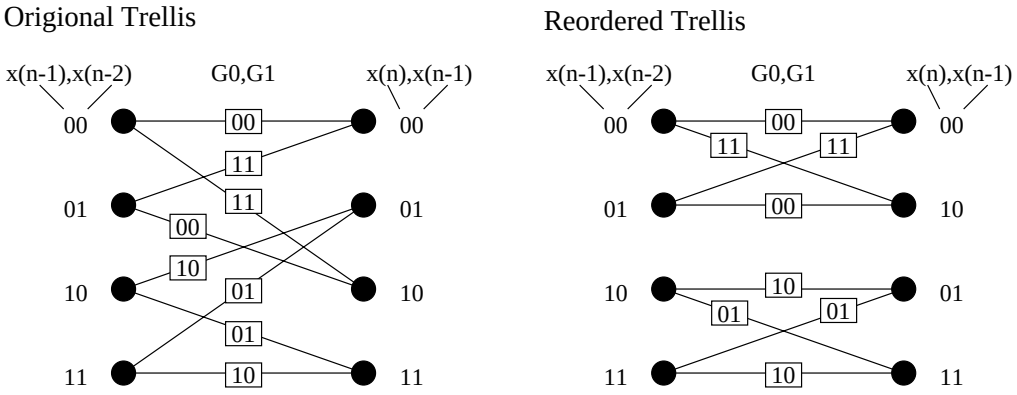

FIGURE 3. Trellis diagrams

There are two trellis diagrams shown in Figure 3. They differ only in the way the destination states are ordered. The purpose for reordering is to obtain higher computational efficiency. The reordered trellis is partitioned into groups of states and state transitions that are isolated from other such groups. This allows realizable, specialized hardware to be designed that is dedicated to compute parameters associated with the particular group. The structure of the groups shown in

Figure 3 is called the “butterfly” due to the physical resemblance. It is also referred to as the radix-2 trellis structure since each group contains 2 source and 2 destination states. The TMS320C54x DSP’s has specialized hardware that is dedicated to compute parameters associated with a butterfly. Note, again, that the structure of the reordered trellis is fixed depending on how the states are defined. Not all trellis diagrams can be reordered into butterflies. Fortunately, a number of commonly used convolution coders have this nice property.

An important feature of this convolution code is that in each butterfly, there are only two (out of four possible) distinct outputs: 00 and 11; or 01 and 10. Notice that the hamming distance of each output pairs is 2. This feature is available to a very small subset of all possible convolution codes. The usefulness of the this feature will be apparent later on when Viterbi decoding is described.

Figure 4 illustrates the entire process of encoding an input pattern to quantizing the received signal to decoding the quantized signal. The sample input pattern used is 1011010100. For the purpose of this example, this input pattern is sufficiently long. In practice, for proper error correction, the input pattern need to be longer than approximately 10 times the number of delay units plus 1, i.e. length of input pattern $= 10(k + 1)$. $(k + 1)$ is often referred to as the constraint length. For this example, k=2 which implies 30 input symbols need to be received before decoding to improve the bit error rate.

The first trellis diagram in Figure 4 illustrates the state transitions associated with each input. This trellis diagram is used here to illustrate how the decoder operates. Use Figure 3 to verify the state transitions and the encoder outputs. The encoder outputs do not have to be generated using the trellis diagram. They can be generated using the system equations directly.

The encoded outputs are transmitted as signed antipodal analog signals (i.e. 0 is transmitted with a positive voltage and 1 is transmitted with a negative voltage). They are received at the decoder and quantized with a 3-bit quantizer. The quantized number is represented in 2’s complement giving it a range of -4 to 3. The process of quantizing a binary analog signal with a multi-bit quantizer is called *soft decision*. In contrast, *hard decision* quantizes the binary analog signal using a 1-bit quantizer (i.e. quantized signal is either 0 or 1). Soft decision offers better performance results since it provides a better estimate of the noise (i.e. less quantization noise is introduced). In most circumstances, the noise is strong enough just to tip the signal over the decision boundary. If hard decision is used, a significant amount of quantization noise will be introduced. The quantized soft decision values are used to calculate the parameters for the Viterbi decoder.

Up to this point, no Viterbi decoding has been performed. Viterbi decoding begins after a certain number of encoded symbols have been received. This length, again, is usually longer than $10(k + 1)$ and is application dependent. In this example, 20 encoded symbols are received before being decoded. In some applications, Viterbi decoding is operated on a frame of received symbols and is independent of the neighboring frames. Is is also possible to perform Viterbi decoding on a sliding window in which a block of decoded bits at the beginning of the window is error free, and the windows will have to overlap. With frame by frame decoding, a number of trailing zeros equalling to the number of states is added. This forces the last state of the frame to be zero providing a starting point for traceback. With sliding window decoding, the starting point for traceback is the state with the optimal accumulated metric. This starting state may be erroneous. However, the traceback path will converge to the correct states before reaching the block of error free bits.

Sample input pattern: 1011010100

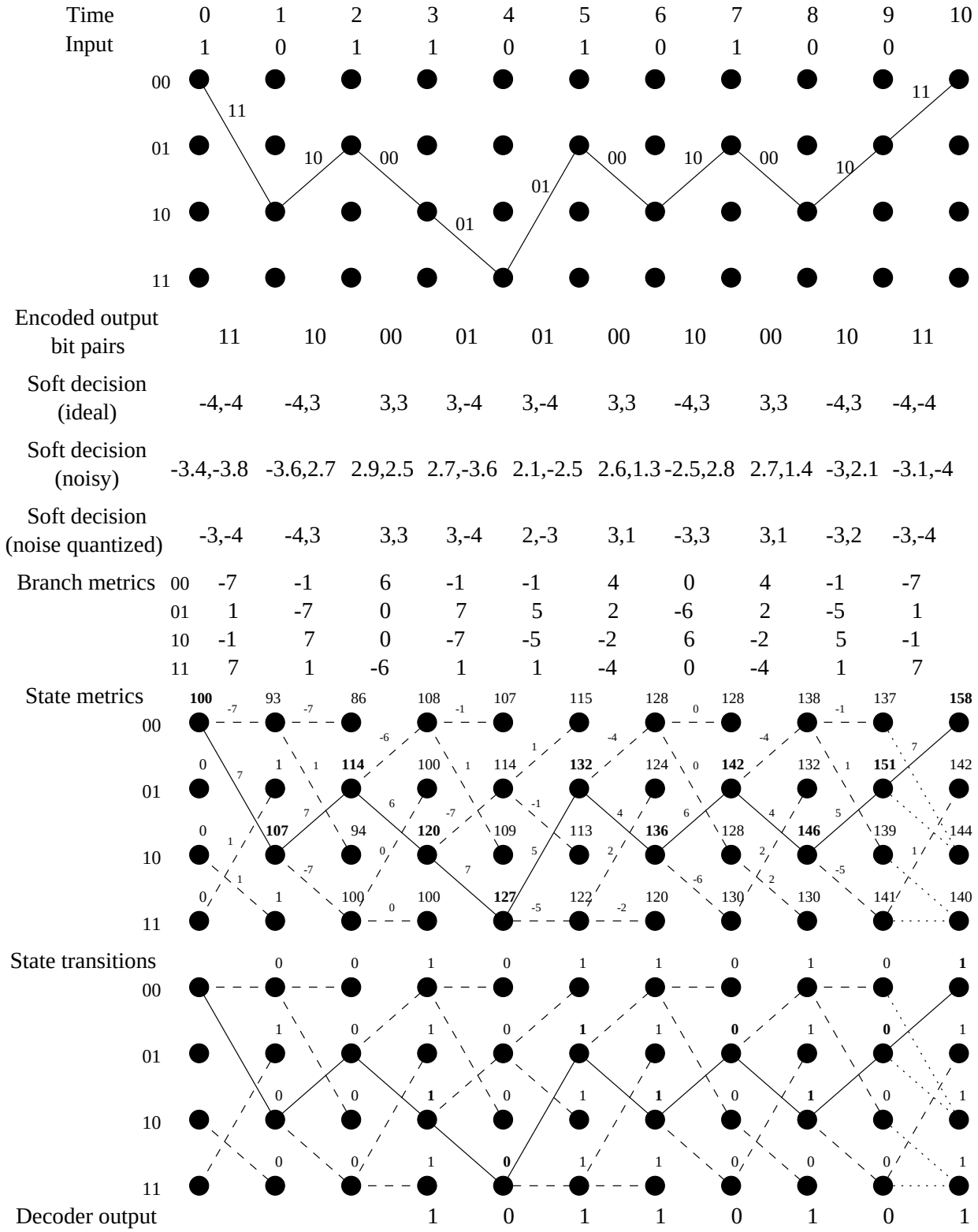

FIGURE 4. Example of convolutional encoding and Viterbi decoding

Viterbi decoding can be broken down into two major operations, metric update and traceback. In metric update, two things are done in every symbol interval (1 symbol = 1 input bit = 2 encoded bits): the accumulated (state) metric is calculated for each state and the optimal incoming path associated with each state is determined. Traceback uses this information to derive an optimal path through the trellis. Referring to the second trellis in Figure 4, notice that a state at any time instance has exactly one incoming path but the number of outgoing paths may vary. This results in a unique path tracing backward.

What are state metrics and how are the optimal incoming paths determined? To understand the metrics used in Viterbi decoding, consider the received encoded bit pairs, $(G_0(n), G_1(n))$. Suppose 11 is transmitted, we would expect 11 to be received with a soft decision pair of (-4,-4). However, channel noise may corrupt the transmitted signal such that the soft decision pair may be (-0.4,-3.3) resulting in a quantized value of (0,-3) (see Figure 5). Without knowing anything about the encoded bit stream, the detector would decide that (3,-4) or 01 is transmitted which results in a bit error. This is called symbol by symbol detection.

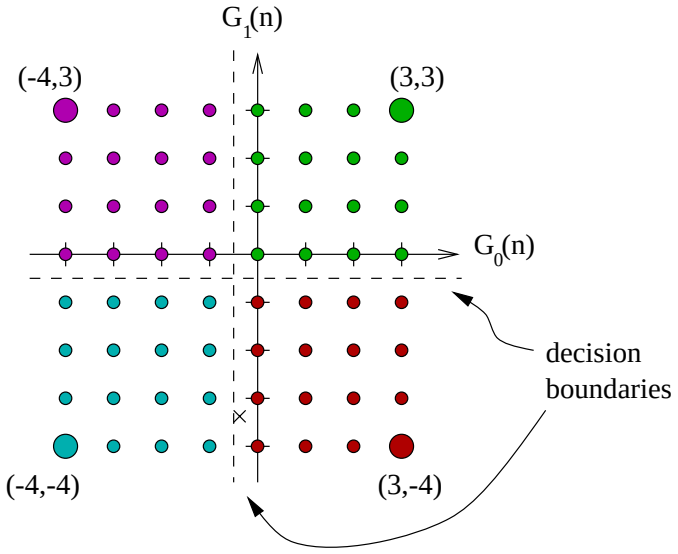

FIGURE 5. Signal constellation for symbol by symbol detection

The Viterbi algorithm, on the other hand, exploits the structure of the convolution code and makes its decision based on previously received data (i.e. this is kept track of using the "states"). Referring back to Figure 3, observe that an encoded bit pair is associated with only 2 possible originating states, $x(n-1), x(n-2)$. For example, if 11 is transmitted, then the originating state must be either state 00 or state 01 (see Figure 3). Suppose that the original state is known to be 00, then it would be impossible for the detected encoded bit pairs to be 01. The Viterbi decoder would then have to decide between the two possible encoded bit pairs, 00 or 11. This decision depends on how far away the received bit pairs are from the two possible transmitted bit pairs (Figure 6). The metric used to measure this is the Euclidean distance. It is also referred to as the local distance calculation.

$$\text{local distance}(n, i) = \sum_{\text{all } j} [S_j(n) - G_j(n)]^2,\ j \in \{\text{encoded bits associated with a given input}\}$$

where n denotes the time instance and i denotes the path calculated for. The above equation can be further simplified by observing the expansion:

$$\text{local distance}(n, i) = \sum_{\text{all } j} [S_j^2(n) - 2S_j(n)G_j(n) + G_j^2(n)]$$

and noting that $\sum_{\text{all } j} S_j^2(n)$ and $\sum_{\text{all } j} G_j^2(n)$ are constants in a given symbol period. These terms can be ignored since we are concerned with finding the minimum local distance path. The following measure will suffice for determining the local distance: Note that the -2 is taken out which implies

$$\text{local distance}_1(n, i) = \sum_{\text{all } j} S_j(n)G_j(n)$$

that instead of finding the path with the minimum local distance, we would look for the path with the maximum $\text{local distance}_1(n, i)$.

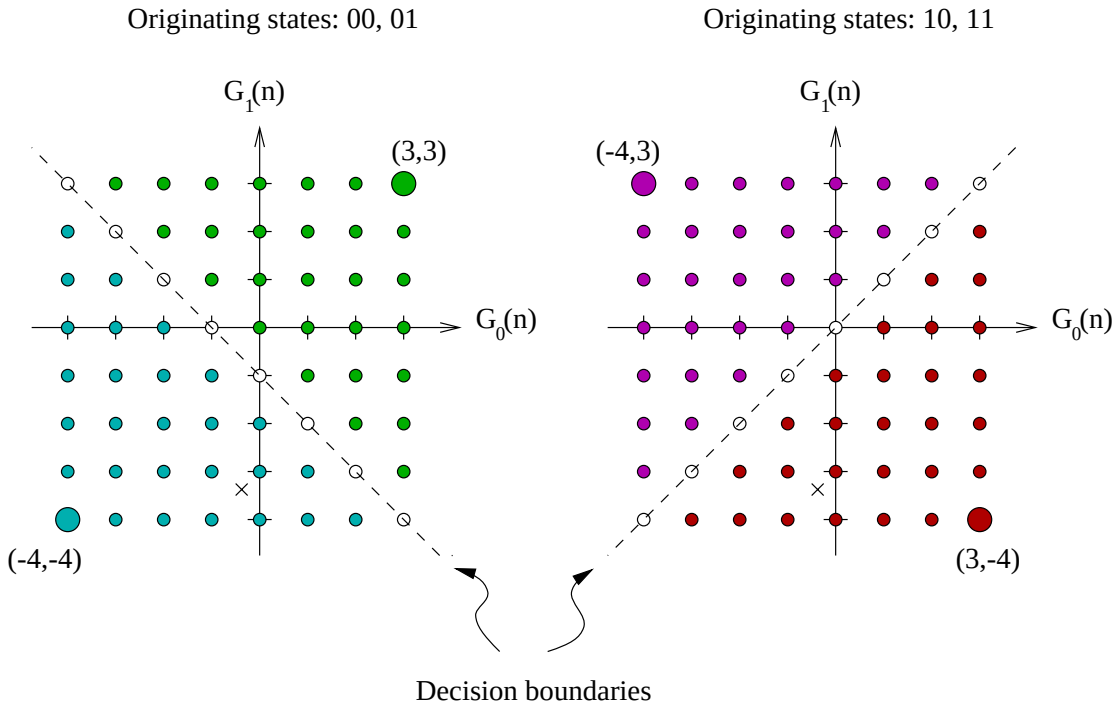

FIGURE 6. Signal constellation for Viterbi decoding

Going back to Figure 4, the branch metrics can now be calculated using the above local distance equation. As an example, the branch metrics for time 0 and 1 is calculated here. The received quantized soft decision pair is (-3,-4). The branch metric corresponding to $(G_0(n), G_1(n)) = (0, 0)$ is equal to $-3(1) - 4(1) = -7$. For $(G_0(n), G_1(n)) = (0, 1)$, it is $-3(1) - 4(-1) = 1$. For $(G_0(n), G_1(n)) = (1, 0)$, it is $-3(-1) - 4(1) = -1$. And for $(G_0(n), G_1(n)) = (1, 1)$, it is $-3(-1) - 4(-1) = 7$. Notice that the $G_j(n)$'s used in the local distance calculation are +1 and -1, rather than +3 and -4 respectively. There are three reasons. First, using unities makes computation very simple. The branch metrics can be calculated by add-

ing and subtracting the soft decision values. Second, 3 and -4 can be scaled to approximately 1 and -1 respectively. As mentioned in the derivation of local distance$_1$, constant scaling can be ignored in determining the maxima and the minima. Finally, note that there are only two unique metric magnitudes, 1 and 7. Therefore, to compute the branch metric, you only need to do 1 add, 1 subtract, and 2 negation.

Once the branch metrics are calculated, the state metrics and the best incoming paths for each destination states can be determined. The state metrics at time 0 are initialized to 0 except for state 00, which takes on the value of 100. This value is arbitrarily chosen and is large enough so that the other initial states cannot contribute to the best path. This basically forces the traceback to converge on state 00 at time 0.

The state metrics are updated in two steps. First, for each of the two incoming paths, the corresponding branch metric is added to the state metric of the originating state. The two sums are compared and the larger one is stored as the new state metric and the corresponding path is stored as the best path. Take state 10 at time instance 2 for example (Figure 4). The two paths coming in to this state originates from states 00 and 01 (Figure 3). Looking at the second trellis, we see that state 00 has a value of 93 and state 01 has a value of 1. The branch metric of the top path (connecting state 00 at time 1 and state 10 at time 2) is 1. The branch metric of the bottom path (between states 01 and 10) is -1. The top path gives a sum of $93 + 1 = 94$ and the bottom path give a sum of $1 + (-1) = 0$. As a result, 94 is stored as the state metric for state 10 at time 2 and the top path is stored as the best path in the transition buffer.

The transition buffer stores the best incoming path for each state. For a radix 2 trellis, only 1 bit is needed to indicate the chosen path. A value of 0 indicates that the top incoming path of the given state is chosen as the best path whereas a value of 1 indicates that the bottom path is chosen. The transition bits for our example is illustrated in Figure 4 in the third trellis.

Traceback begins after completing the metric update of the last symbol in the frame. For frame by frame Viterbi decoding, all that is needed by the traceback algorithm are the state transitions. The starting state is state 00. For sliding window decoding, the starting state is the state with the largest state metric. In our example, the starting state for both types of decoding is 00 since it has the largest state metric, 158. Looking at the state transition for state 00, we see that the bottom path is optimal (state transition = 1 implies bottom path). Two things now happen. First, the transition bit, 1, is sent to the decoder output. Second, the originating state, state 01, of this optimal path is determined. This process is repeated until time 2, which corresponds to the first input bit transmitted.

The decoded output is shown in Figure 4 in the last row. Notice that the order with which the decoded output is generated is reversed, i.e. decoded output = 10101101 whereas the sample input pattern is 10110101. Additional code need to be added to reverse the ordering of the decoded output to exactly reconstruct the sample input.