Chapter 2. Convolutional Codes

This chapter describes the encoder and decoder structures for convolutional codes. The encoder will be represented in many different but equivalent ways. Also, the main decoding strategy for convolutional codes, based on the Viterbi Algorithm, will be described. A firm understanding of convolutional codes is an important prerequisite to the understanding of turbo codes.

2.1 Encoder Structure

A convolutional code introduces redundant bits into the data stream through the use of linear shift registers as shown in Figure 2.1.

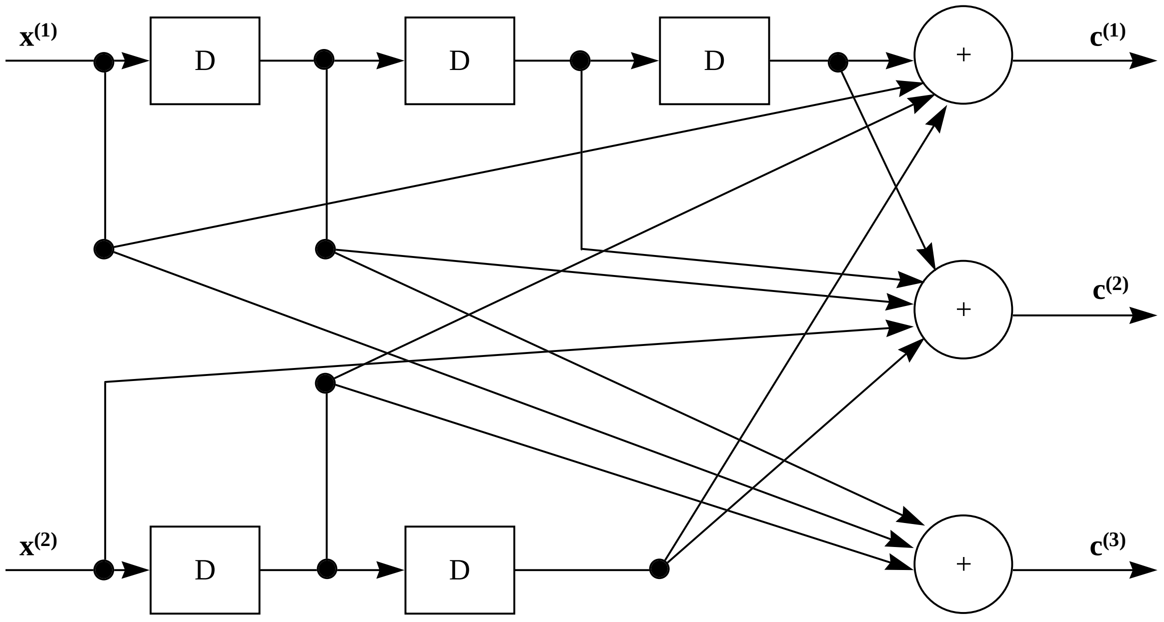

Figure 2.1: Example convolutional encoder where $x^{(i)}$ is an input information bit stream and $c^{(i)}$ is an output encoded bit stream [Wic95].

The information bits are input into shift registers and the output encoded bits are obtained by modulo-2 addition of the input information bits and the contents of the shift registers. The connections to the modulo-2 adders were developed heuristically with no algebraic or combinatorial foundation.

The code rate r for a convolutional code is defined as

$$r = \frac{k}{n} \tag{2.1}$$

where k is the number of parallel input information bits and n is the number of parallel output encoded bits at one time interval. The constraint length K for a convolutional code is defined as

$$K = m + 1 \tag{2.2}$$

where m is the maximum number of stages (memory size) in any shift register. The shift registers store the state information of the convolutional encoder and the constraint length relates the number of bits upon which the output depends. For the convolutional encoder shown in Figure 2.1, the code rate r=2/3, the maximum memory size m=3, and the constraint length K=4.

A convolutional code can become very complicated with various code rates and constraint lengths. As a result, a simple convolutional code will be used to describe the code properties as shown in Figure 2.2.

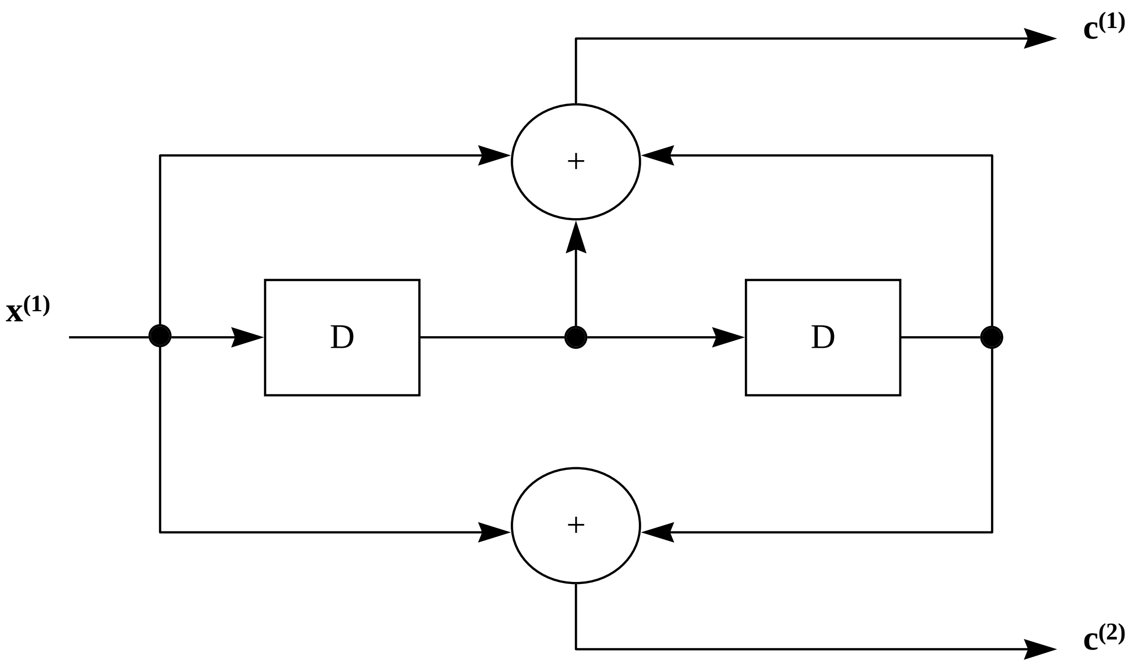

Figure 2.2: Convolutional encoder with k=1, n=2, r=1/2, m=2, and K=3.

2.2 Encoder Representations

The encoder can be represented in several different but equivalent ways. They are

1. Generator Representation
2. Tree Diagram Representation
3. State Diagram Representation
4. Trellis Diagram Representation

2.2.1 Generator Representation

Generator representation shows the hardware connection of the shift register taps to the modulo-2 adders. A generator vector represents the position of the taps for an output. A “1” represents a connection and a “0” represents no connection. For example, the two generator vectors for the encoder in Figure 2.2 are $\mathbf{g_1}$ = [111] and $\mathbf{g_2}$ = [101] where the subscripts 1 and 2 denote the corresponding output terminals.

2.2.2 Tree Diagram Representation

The tree diagram representation shows all possible information and encoded sequences for the convolutional encoder. Figure 2.3 shows the tree diagram for the encoder in Figure 2.2 for four input bit intervals.

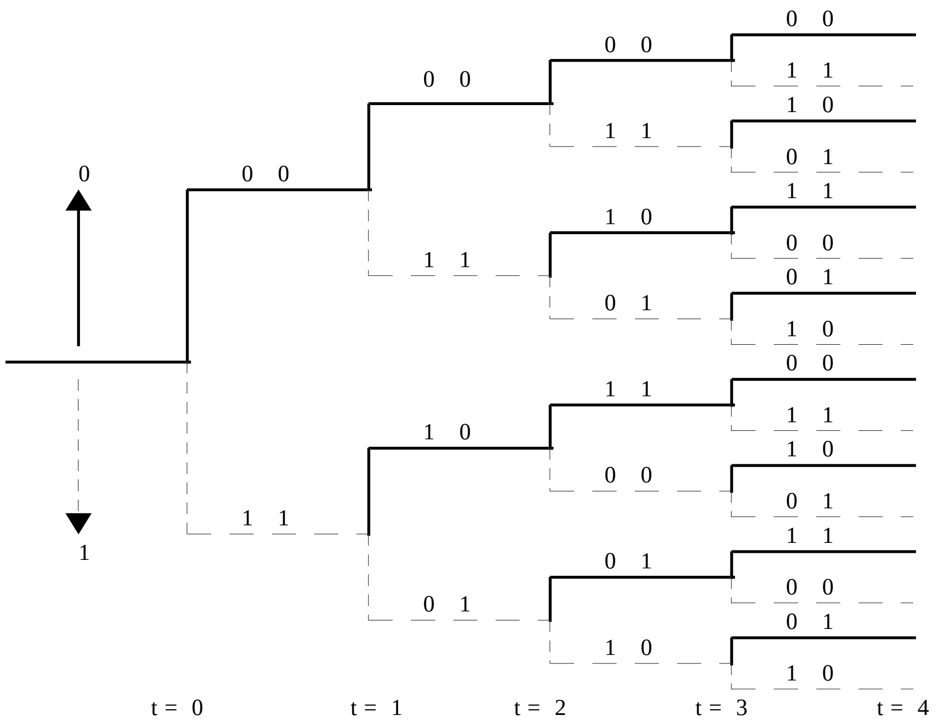

Figure 2.3: Tree diagram representation of the encoder in Figure 2.2 for four input bit intervals.

In the tree diagram, a solid line represents input information bit 0 and a dashed line represents input information bit 1. The corresponding output encoded bits are shown on the branches of the tree. An input information sequence defines a specific path through the tree diagram from left to right. For example, the input information sequence

x={1011} produces the output encoded sequence **c**={11, 10, 00, 01}. Each input information bit corresponds to branching either upward (for input information bit 0) or downward (for input information bit 1) at a tree node.

2.2.3 State Diagram Representation

The state diagram shows the state information of a convolutional encoder. The state information of a convolutional encoder is stored in the shift registers. Figure 2.4 shows the state diagram of the encoder in Figure 2.2.

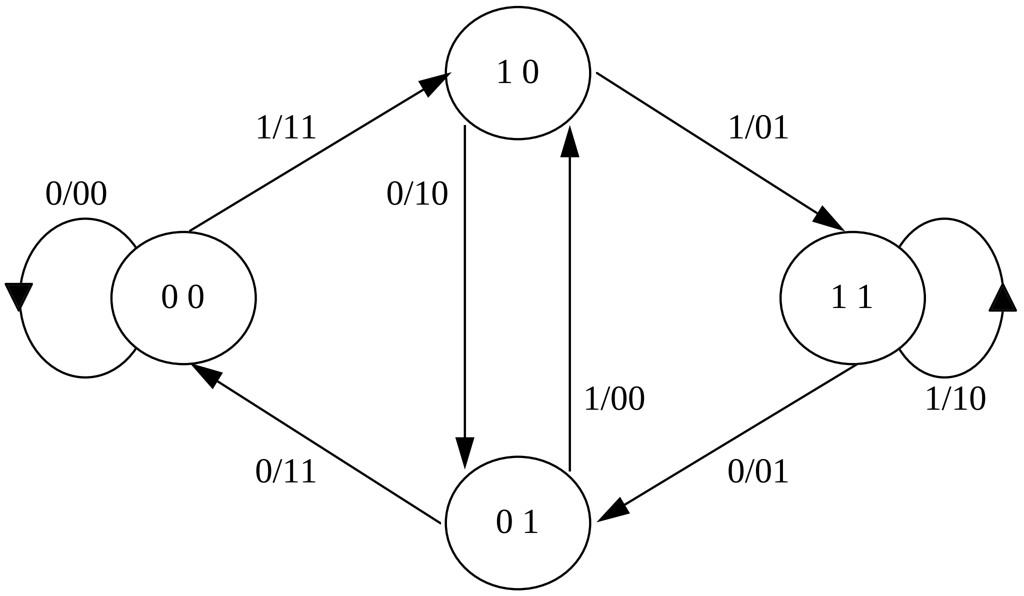

Figure 2.4: State diagram representation of the encoder in Figure 2.2.

In the state diagram, the state information of the encoder is shown in the circles. Each new input information bit causes a transition from one state to another. The path information between the states, denoted as x/**c**, represents input information bit x and output encoded bits **c**. It is customary to begin convolutional encoding from the all zero state. For example, the input information sequence **x**={1011} (begin from the all zero state) leads to the state transition sequence s={10, 01, 10, 11} and produces the output encoded sequence c={11, 10, 00, 01}. Figure 2.5 shows the path taken through the state diagram for the given example.

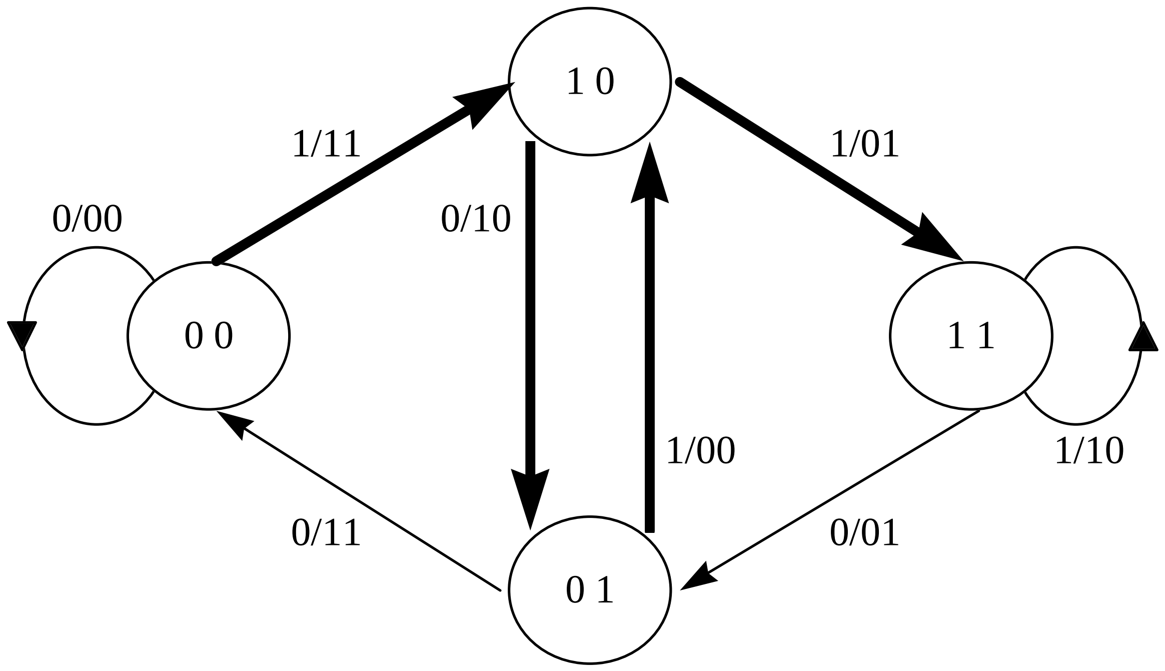

Figure 2.5: The state transitions (path) for input information sequence {1011}.

2.2.4 Trellis Diagram Representation

The trellis diagram is basically a redrawing of the state diagram. It shows all possible state transitions at each time step. Frequently, a legend accompanies the trellis diagram to show the state transitions and the corresponding input and output bit mappings (x/**c**). This compact representation is very helpful for decoding convolutional codes as discussed later. Figure 2.6 shows the trellis diagram for the encoder in Figure 2.2.

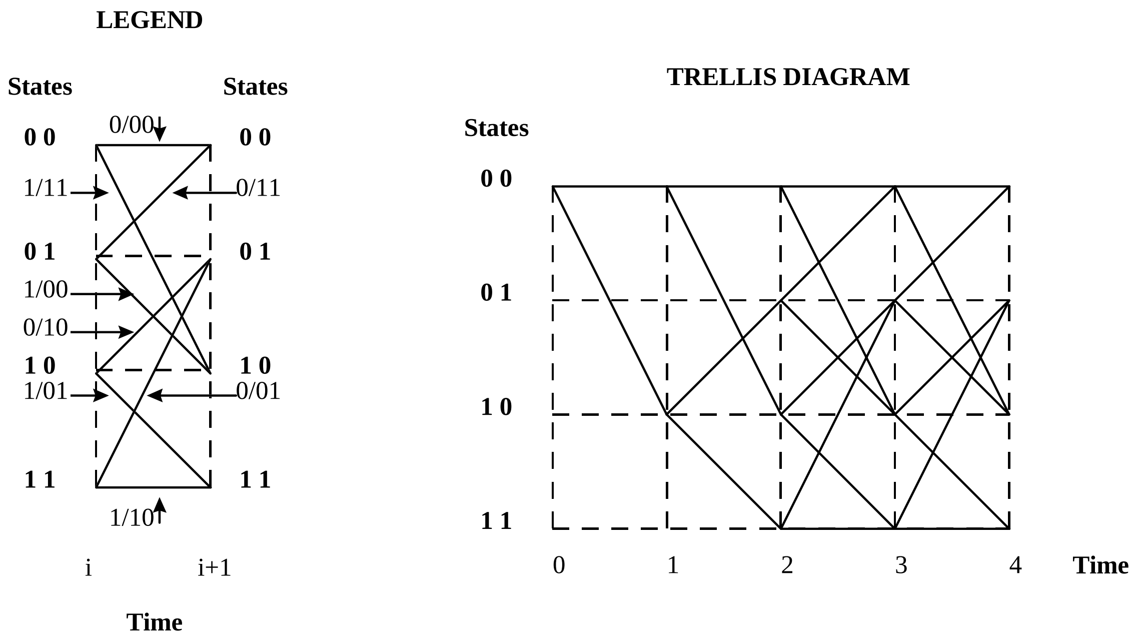

Figure 2.6: Trellis diagram representation of the encoder in Figure 2.2 for four input bit intervals.

Figure 2.7 shows the trellis path for the state transitions in Figure 2.5.

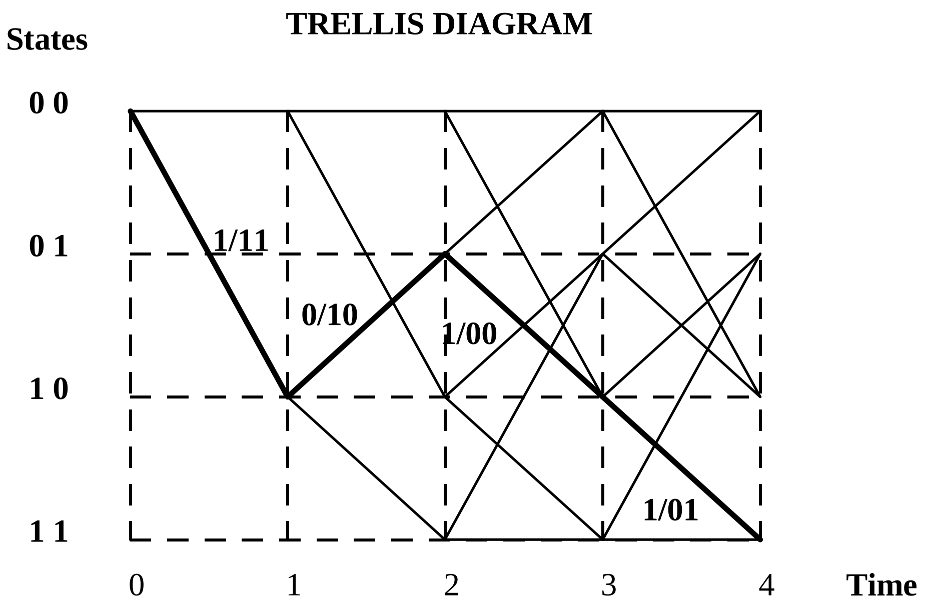

Figure 2.7: Trellis path for the state transitions in Figure 2.5.

2.3 Catastrophic Convolutional Code

Catastrophic convolutional code causes a large number of bit errors when only a small number of channel bit errors is received. This type of code needs to be avoided and can be identified by the state diagram. A state diagram having a loop in which a nonzero information sequence corresponds to an all-zero output sequence identifies a catastrophic convolutional code. Figure 2.8 shows two examples of such code.

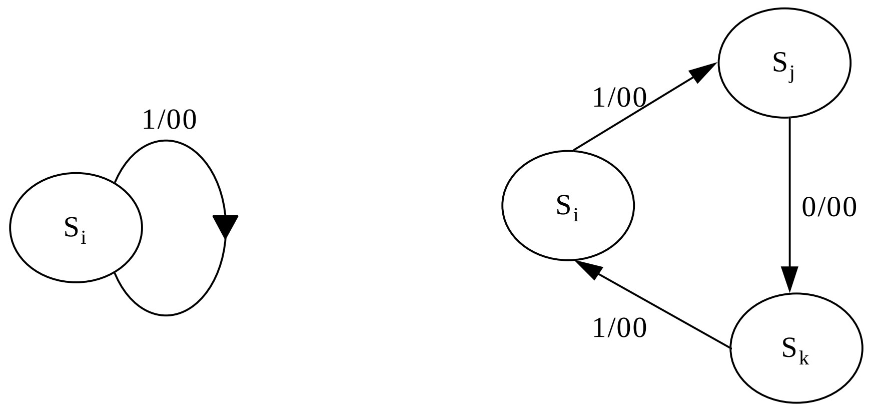

Figure 2.8: Examples of catastrophic convolutional code.

2.4 Hard-Decision and Soft-Decision Decoding

Hard-decision and soft-decision decoding refer to the type of quantization used on the received bits. Hard-decision decoding uses 1-bit quantization on the received channel values. Soft-decision decoding uses multi-bit quantization on the received channel values. For the ideal soft-decision decoding (infinite-bit quantization), the received channel values are directly used in the channel decoder. Figure 2.9 shows hard- and soft-decision decoding.

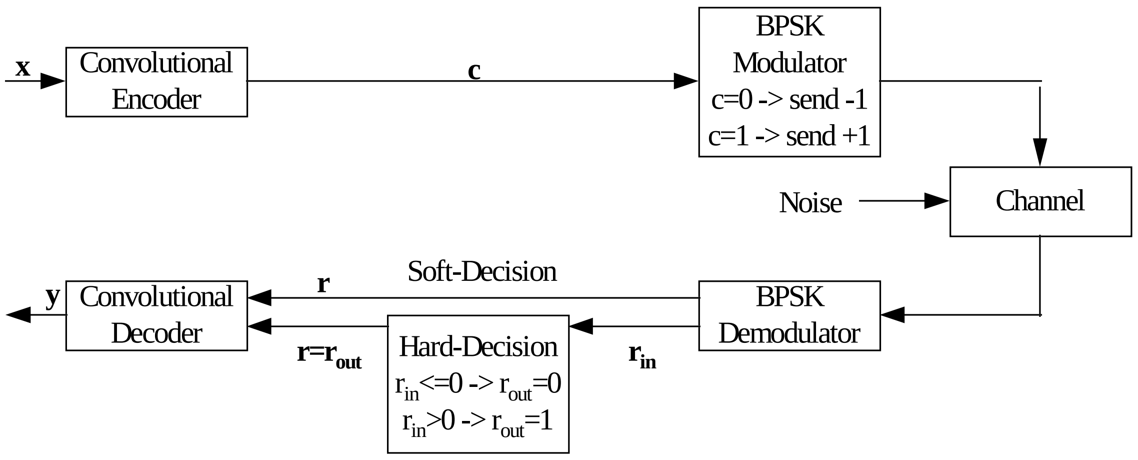

Figure 2.9: Hard- and Soft-decision decoding [Woe94].

2.5 Hard-Decision Viterbi Algorithm

For a convolutional code, the input sequence **x** is “convoluted” to the encoded sequence **c.** Sequence **c** is transmitted across a noisy channel and the received sequence **r** is obtained. The Viterbi algorithm computes a maximum likelihood (ML) estimate on the estimated code sequence **y** from the received sequence **r** such that it maximizes the probability p(**r**|**y**) that sequence **r** is received conditioned on the estimated code sequence **y.** Sequence **y** must be one of the allowable code sequences and cannot be any arbitrary sequence. Figure 2.10 shows the described system structure.

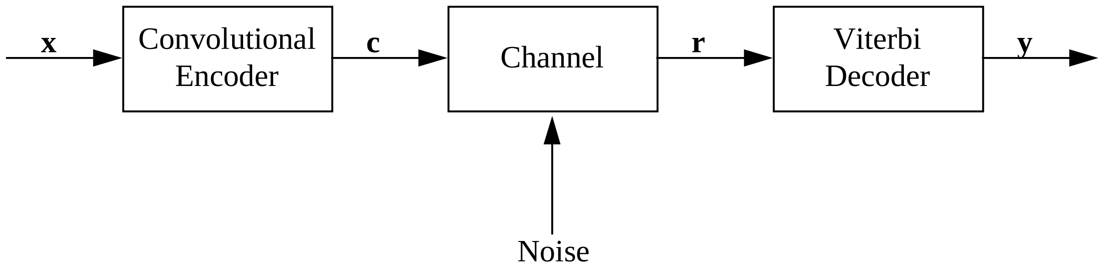

Figure 2.10: Convolutional code system.

For a rate r convolutional code, the encoder inputs k bits in parallel and outputs n bits in parallel at each time step. The input sequence is denoted as

$$\mathbf{x}=(x_0^{(1)}, x_0^{(2)}, ..., x_0^{(k)}, x_1^{(1)}, ..., x_1^{(k)}, x_{L+m-1}^{(1)}, ..., x_{L+m-1}^{(k)}) \qquad (2.3)$$

and the coded sequence is denoted as

$$\mathbf{c}=(c_0^{(1)}, c_0^{(2)}, ..., c_0^{(n)}, c_1^{(1)}, ..., c_1^{(n)}, c_{L+m-1}^{(1)}, ..., c_{L+m-1}^{(n)}) \qquad (2.4)$$

where L denotes the length of input information sequence and m denotes the maximum length of the shift registers. Additional m zero bits are required at the tail of the information sequence to take the convolutional encoder back to the all-zero state. It is required that the encoder start and end at the all-zero state. The subscript denotes the time index while the superscript denotes the bit within a particular input k-bit or output n-bit block. The received and estimated sequences **r** and **y** can be described similarly as

$$\mathbf{r}=(r_0^{(1)},\ r_0^{(2)}, ..., r_0^{(n)}, r_1^{(1)}, ..., r_1^{(n)}, r_{L+m-1}^{(1)},\ ..., r_{L+m-1}^{(n)}) \tag{2.5}$$

and

$$\mathbf{y}=(y_0^{(1)}, y_0^{(2)}, ..., y_0^{(n)}, y_1^{(1)}, ..., y_1^{(n)}, y_{L+m-1}^{(1)},\ ..., y_{L+m-1}^{(n)}). \tag{2.6}$$

For ML decoding, the Viterbi algorithm selects **y** to maximize p(**r**|**y**). The channel is assumed to be memoryless, and thus the noise process affecting a received bit is independent from the noise process affecting all of the other received bits [Wic95]. From probability theory, the probability of joint, independent events is equivalent to the product of the probabilities of the individual events. Thus,

$$p(\mathbf{r}|\mathbf{y}) = \prod_{i=0}^{L+m-1}[\,p(r_i^{(1)}|y_i^{(1)})\,p(r_i^{(2)}|y_i^{(2)})\cdots p(r_i^{(n)}|y_i^{(n)})] \text{ [Wic95]} \tag{2.7}$$

$$= \prod_{i=0}^{L+m-1}\left(\prod_{j=1}^{n} p(r_i^{(j)}|y_i^{(j)})\right) \text{ [Wic95]} \tag{2.8}$$

This equation is called the likelihood function of **y** given that **r** is received [Vit71]. The estimate that maximizes p(**r**|**y**) also maximizes log p(**r**|**y**) because logarithms are monotonically increasing functions. Thus, a log likelihood function can be defined as

$$\log p(\mathbf{r}|\mathbf{y}) = \sum_{i=0}^{L+m-1}\left(\sum_{j=1}^{n}\log p(r_i^{(j)}|y_i^{(j)})\right) \text{ [Wic95]} \tag{2.9}$$

For an easier manipulation of the summations over the log function, a bit metric is defined. The bit metric is defined as

$$M(r_i^{(j)}|y_i^{(j)}) = a[\log p(r_i^{(j)}|y_i^{(j)}) + b] \text{ [Wic95]} \tag{2.10}$$

where a and b are chosen such that the bit metric is a small positive integer [Wic95]. The values *a* and *b* are defined for binary symmetric channel (BSC) or hard-decision decoding. Figure 2.11 shows a BSC.

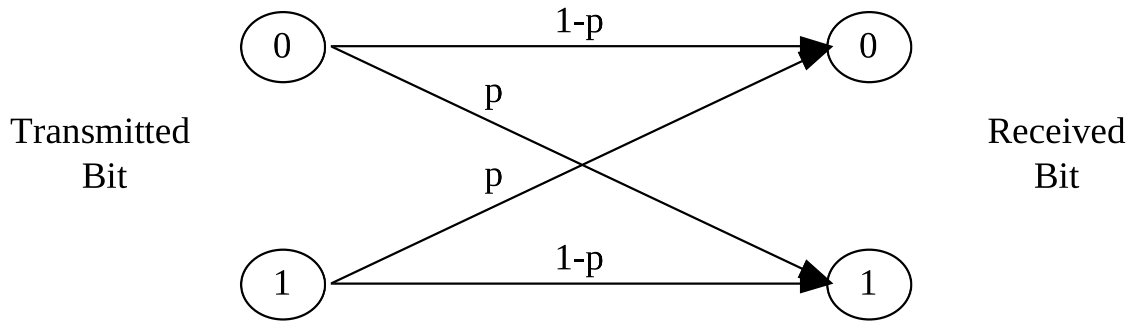

Figure 2.11: The binary symmetric channel model, where p is the crossover probability.

For BSC, *a* and *b* can be chosen in two distinct ways. For the conventional way, they can be chosen as

$$a = \frac{1}{\log p - \log(1-p)} \quad \text{[Wic95]} \tag{2.11}$$

and

$$b = -\log(1-p) \quad \text{[Wic95]} \tag{2.12}$$

The resulting bit metric is then

$$M(r_i^{(j)}|y_i^{(j)}) = \frac{1}{[\log p - \log(1-p)]}[\log p(r_i^{(j)}|y_i^{(j)}) - \log(1-p)] \tag{2.13}$$

From the BSC model, it is clear that $p(r_i^{(j)}|y_i^{(j)})$ can only take on values *p* and 1-*p*. Table 2.1 shows the resulting bit metric.

Table 2.1: Conventional Bit Metric Values

$M(r_i^{(j)}\|y_i^{(j)})$	**Received Bit** $r_i^{(j)} = 0$	**Received Bit** $r_i^{(j)} = 1$
Decoded Bit $y_i^{(j)} = 0$	0	1
Decoded Bit $y_i^{(j)} = 1$	1	0

This bit metric shows the cost of receiving and decoding bits. For example, if the decoded bit $y_i^{(j)} = 0$ and the received bit $r_i^{(j)} = 0$, then the cost $M(r_i^{(j)}|y_i^{(j)})$ = 0. However, if the decoded bit $y_i^{(j)} = 0$ and the received bit $r_i^{(j)} = 1$, then the cost $M(r_i^{(j)}|y_i^{(j)})$ = 1. As it can be seen, this is related to the Hamming distance and is known as the Hamming distance metric. Thus, the Viterbi algorithm chooses the code sequence **y** through the trellis that has the smallest cost/Hamming distance relative to the received sequence **r**.

Alternatively, a and b can be chosen as

$$a = \frac{1}{\log(1-p) - \log p} \quad \text{[Wic95]} \tag{2.14}$$

and

$$b = -\log p \quad \text{[Wic95]} \tag{2.15}$$

The resulting alternative bit metric is then

$$M(r_i^{(j)}|y_i^{(j)}) = \frac{1}{[\log(1-p) - \log p]}[\log p(r_i^{(j)}|y_i^{(j)}) - \log p] \tag{2.16}$$

Table 2.2 shows the resulting alternative bit metric.

Table 2.2: Alternative Bit Metric Values

$M(r_i^{(j)}\|y_i^{(j)})$	**Received Bit** $r_i^{(j)} = 0$	**Received Bit** $r_i^{(j)} = 1$
Decoded Bit $y_i^{(j)} = 0$	1	0
Decoded Bit $y_i^{(j)} = 1$	0	1

For this case, the Viterbi algorithm chooses the code sequence **y** through the trellis that has the largest cost/Hamming distance relative to the received sequence **r**. Furthermore, for an arbitrary channel (not necessarily BSC), the values a and b are found on a trial-and-error basis to obtain an acceptable bit metric.

From the bit metric, a path metric is defined. The path metric is defined as

$$M(\mathbf{r}|\mathbf{y}) = \sum_{i=0}^{L+m-1}\left(\sum_{j=1}^{n} M(r_i^{(j)}|y_i^{(j)})\right) \quad \text{[Wic95]} \tag{2.17}$$

and indicates the total cost of estimating the received bit sequence **r** with the decoded bit sequence **y** in the trellis diagram. Furthermore, the kth branch metric is defined as

$$M(\mathbf{r}_k|\mathbf{y}_k) = \sum_{j=1}^{n} M(r_k^{(j)}|y_k^{(j)}) \quad \text{[Wic95]} \tag{2.18}$$

and the kth partial path metric is defined as

$$M^k(\mathbf{r}|\mathbf{y}) = \sum_{i=0}^{k} M(\mathbf{r}_i|\mathbf{y}_i) \quad \text{[Wic95]} \tag{2.19}$$

$$= \sum_{i=0}^{k}\left(\sum_{j=1}^{n} M(r_i^{(j)}|y_i^{(j)})\right) \quad \text{[Wic95]} \tag{2.20}$$

The kth branch metric indicates the cost of choosing a branch from the trellis diagram. The kth partial path metric indicates the cost of choosing a partially decoded bit sequence **y** up to time index k.

The Viterbi algorithm utilizes the trellis diagram to compute the path metrics. Each state (node) in the trellis diagram is assigned a value, the partial path metric. The partial path metric is determined from state s = 0 at time t = 0 to a particular state s = k at time $t \geq 0$. At each state, the "best" partial path metric is chosen from the paths terminated at that state [Wic95]. The "best" partial path metric may be either the larger or smaller metric, depending whether *a* and *b* are chosen conventionally or alternatively. The selected metric represents the survivor path and the remaining metrics represent the nonsurvivor paths. The survivor paths are stored while the nonsurvivor paths are discarded in the trellis diagram. The Viterbi algorithm selects the single survivor path left at the end of the process as the ML path. Trace-back of the ML path on the trellis diagram would then provide the ML decoded sequence.

The hard-decision Viterbi algorithm (HDVA) can be implemented as follows [Rap96], [Wic95]:

$S_{k,t}$ is the state in the trellis diagram that corresponds to state S_k at time t. Every state in the trellis is assigned a value denoted $V(S_{k,t})$.

1. (a) Initialize time t = 0.
   (b) Initialize $V(S_{0,0}) = 0$ and all other $V(S_{k,t}) = +\infty$.
2. (a) Set time t = t+1.
   (b) Compute the partial path metrics for all paths going to state S_k at time t.
   First, find the t^th branch metric $M(\mathbf{r}_t|\mathbf{y}_t) = \sum_{j=1}^{n} M(r_t^{(j)}|y_t^{(j)})$. This is calculated from the Hamming distance $\sum_{j=1}^{n} \left| r_t^{(j)} - y_t^{(j)} \right|$. Second, compute the t^th partial path metric $M^t(\mathbf{r}|\mathbf{y}) = \sum_{i=0}^{t} M(\mathbf{r}_i|\mathbf{y}_i)$. This is calculated from $V(S_{k,t-1}) + M(\mathbf{r}_t|\mathbf{y}_t)$.
3. (a) Set $V(S_{k,t})$ to the "best" partial path metric going to state S_k at time t. Conventionally, the "best" partial path metric is the partial path metric with the smallest value.
   (b) If there is a tie for the "best" partial path metric, then any one of the tied partial path metric may be chosen.
4. Store the "best" partial path metric and its associated survivor bit and state paths.
5. If t < L+m-1, return to Step 2.

The result of the Viterbi algorithm is a unique trellis path that corresponds to the ML codeword.

A simple HDVA decoding example is shown below. The convolutional encoder used is shown in Figure 2.2. The input sequence is **x**={1010100}, where the last two bits are used to return the encoder to the all-zero state. The coded sequence is **c**={1<u>1</u>, 10, 00, 10, 00, 10, 11}. However, the received sequence **r**={1<u>0</u>, 10, 00, 10, 00, 10, 11} has a bit error (underlined). Figure 2.12 shows the state transition diagram (trellis legend) of the example convolutional encoder.

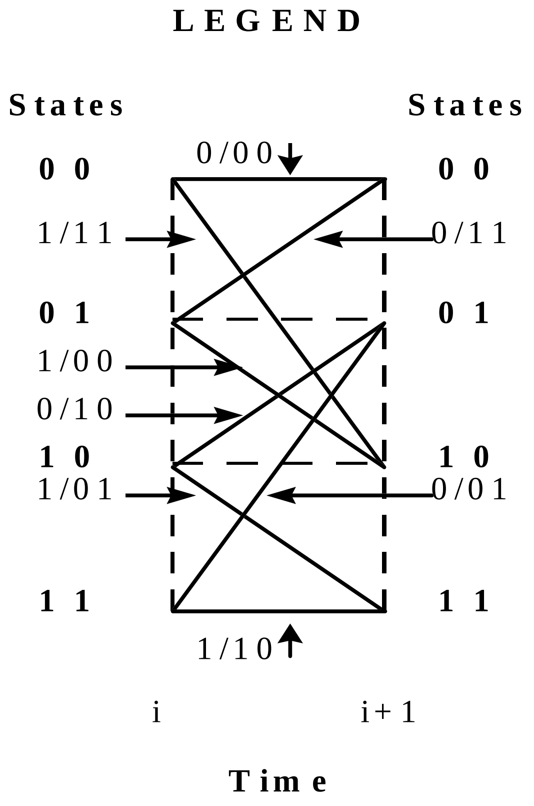

Figure 2.12: The state transition diagram (trellis legend) of the example convolutional encoder.

The state transition diagram shows the estimated information and coded bits along the branches (needed for the decoding process). HDVA decoding chooses the ML path through the trellis as shown in Figure 2.13. The chosen partial path (accumulated) metric for this example is the smallest Hamming distance and are shown in the figure for every node. The bold partial path metrics correspond to the ML path. Survivor paths are represented by bold solid lines and competing paths are represented by simple solid lines. For metric "ties", the first branch is always chosen.

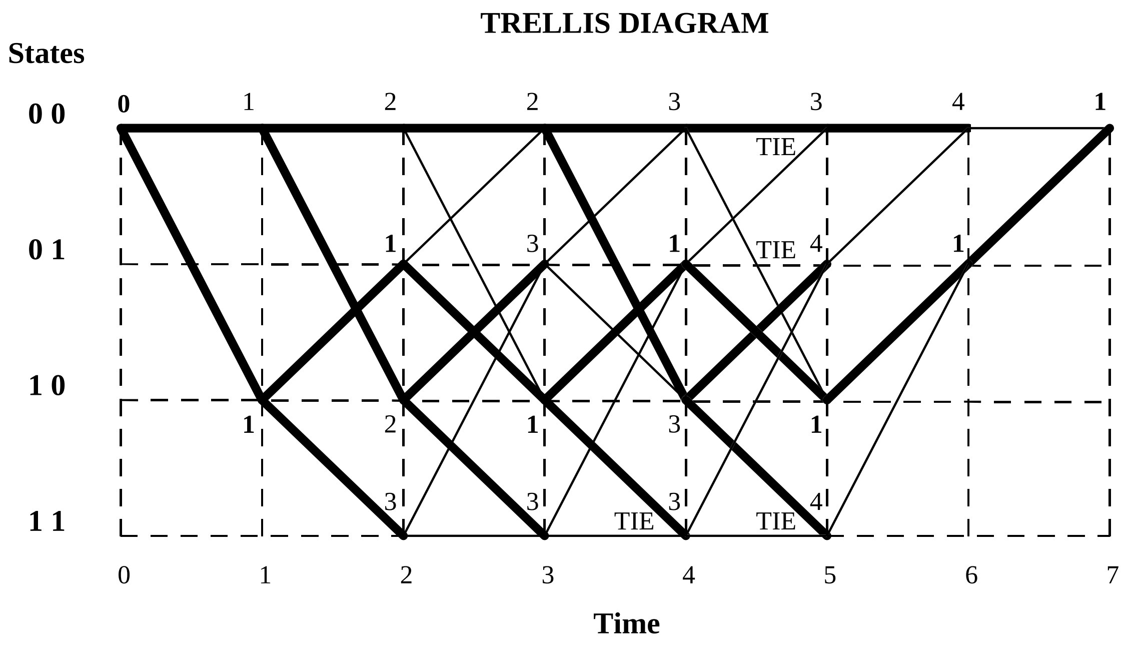

Figure 2.13: HDVA decoding of the example.

From the trellis diagram in Figure 2.13, the estimated code sequence is **y**={11, 10, 00, 10, 00, 10, 11} which is the code sequence **c**. Utilizing the state transition diagram in Figure 2.12, the estimated information sequence is **x'**={1010100}.

2.6 Soft-Decision Viterbi Algorithm

There are two general methods of implementing a soft-decision Viterbi algorithm. The first method (Method 1) uses Euclidean distance metric instead of Hamming distance metric. The received bits used in the Euclidean distance metric are processed by multi-bit quantization. The second method (Method 2) uses a correlation metric where its received bits used in this metric are also processed by multi-bit quantization.

2.6.1 Soft-Decision Viterbi Algorithm (Method 1)

In soft-decision decoding, the receiver does not assign a zero or a one (single-bit quantization) to each received bit but uses multi-bit or infinite-bit quantized values [Wic95]. Ideally, the received sequence **r** is infinite-bit quantized and is used directly in the soft-decision Viterbi decoder. The soft-decision Viterbi algorithm is similar to its hard-decision algorithm except that squared Euclidean distance is used in the metric instead of Hamming distance.

The soft-decision Viterbi algorithm (SDVA1) can be implemented as follows:

$S_{k,t}$ is the state in the trellis diagram that corresponds to state S_k at time t. Every state in the trellis is assigned a value denoted $V(S_{k,t})$.

1. (a) Initialize time t = 0.
   (b) Initialize $V(S_{0,0}) = 0$ and all other $V(S_{k,t}) = +\infty$.
2. (a) Set time t = t+1.
   (b) Compute the partial path metrics for all paths going to state S_k at time t. First, find the t^th branch metric $M(\mathbf{r}_t|\mathbf{y}_t) = \sum_{j=1}^{n} M(r_t^{(j)}|y_t^{(j)})$. This is calculated from the squared Euclidean distance $\sum_{j=1}^{n}(r_t^{(j)} - y_t^{(j)})^2$. Second, compute the t^th partial path metric $M^t(\mathbf{r}|\mathbf{y}) = \sum_{i=0}^{t} M(\mathbf{r}_i|\mathbf{y}_i)$. This is calculated from $\mathrm{V(S_{k,t\text{-}1})} + M(\mathbf{r}_t|\mathbf{y}_t)$.
3. (a) Set $V(S_{k,t})$ to the "best" partial path metric going to state S_k at time t. Conventionally, the "best" partial path metric is the partial path metric with the smallest value.
   (b) If there is a tie for the "best" partial path metric, then any one of the tied partial path metric may be chosen.
4. Store the "best" partial path metric and its associated survivor bit and state paths.
5. If t < L+m-1, return to Step 2.

2.6.2 Soft-Decision Viterbi Algorithm (Method 2)

The second soft-decision Viterbi algorithm (SDVA2) is developed below. The likelihood function is represented by a Gaussian probability density function

$$p(r_i^{(j)}|y_i^{(j)}) = \frac{1}{\sqrt{\pi N_o}} e^{-(r_i^{(j)} - y_i^{(j)}\sqrt{E_b})^2/N_o} \tag{2.21}$$

where E_b is the received energy per code-word bit and N_o is the one-sided noise spectral density [Wic95]. The received bit is a Gaussian random variable with mean $y_i^{(j)}\sqrt{E_b}$ and variance $N_o/2$. The log likelihood function can be defined as [Wic95]

$$\log p(\mathbf{r}|\mathbf{y}) = \sum_{i=0}^{L+m-1}\left(\sum_{j=1}^{n}\log p(r_i^{(j)}|y_i^{(j)})\right) \tag{2.22}$$

$$= \sum_{i=0}^{L+m-1}\left\{\sum_{j=1}^{n}\left[-\frac{(r_i^{(j)} - y_i^{(j)}\sqrt{E_b})^2}{N_o} - \log\sqrt{\pi N_o}\right]\right\} \tag{2.23}$$

$$= \frac{-1}{N_o}\sum_{i=0}^{L+m-1}\left[\sum_{j=1}^{n}(r_i^{(j)} - y_i^{(j)}\sqrt{E_b})^2\right] - \frac{(L+m)n}{2}\log \pi N_o \tag{2.24}$$

$$= \frac{-1}{N_o} \sum_{i=0}^{L+m-1} \left\{ \sum_{j=1}^{n} [r_i^{(j)2} - 2r_i^{(j)} y_i^{(j)} \sqrt{E_b} + y_i^{(j)2} E_b] \right\} - \frac{(L+m)n}{2} \log \pi N_o \qquad (2.25)$$

where $y_i^{(j)2} = 1$

$$= C_1 \sum_{i=0}^{L+m-1} \left[\sum_{j=1}^{n} r_i^{(j)} y_i^{(j)} \right] + C_2 \qquad (2.26)$$

where C_1 and C_2 are all terms not a function of **y**

$$= C_1 (\mathbf{r} \bullet \mathbf{y}) + C_2 \qquad (2.27)$$

From this, it is seen that the bit metric can be defined as

$$M(r_i^{(j)} | y_i^{(j)}) = r_i^{(j)} y_i^{(j)} \qquad (2.28)$$

The soft-decision Viterbi algorithm (SDVA2) can be implemented as follows:

$S_{k,t}$ is the state in the trellis diagram that corresponds to state S_k at time t. Every state in the trellis is assigned a value denoted $V(S_{k,t})$.

1. (a) Initialize time t = 0.
   (b) Initial $V(S_{0,0}) = 0$ and all other $V(S_{k,t}) = -\infty$.
2. (a) Set time t = t+1.
   (b) Compute the partial path metrics for all paths going to state S_k at time t. First, find the tth branch metric $M(\mathbf{r}_t | \mathbf{y}_t) = \sum_{j=1}^{n} M(r_t^{(j)} | y_t^{(j)})$. This is calculated from the correlation of $r_i^{(j)}$ and $y_i^{(j)}$, $\sum_{j=1}^{n} r_i^{(j)} y_i^{(j)}$. Second, compute the tth partial path metric $M^t(\mathbf{r} | \mathbf{y}) = \sum_{i=0}^{t} M(\mathbf{r}_i | \mathbf{y}_i)$. This is calculated from $V(S_{k,t-1}) + M(\mathbf{r}_t | \mathbf{y}_t)$.
3. (a) Set $V(S_{k,t})$ to the “best” partial path metric going to state S_k at time t. The “best” partial path metric is the partial path metric with the largest value.
   (b) If there is a tie for the “best” partial path metric, then any one of the tied partial path metric may be chosen.
4. Store the “best” partial path metric and its associated survivor bit and state paths.
5. If t < L+m-1, return to Step 2.

Generally with soft-decision decoding, approximately 2 dB of coding gain over hard-decision decoding can be obtained in Gaussian channels.

2.7 Performance Analysis of Convolutional Code

The performance of convolutional codes can be quantified through analytical means or by computer simulation. The analytical approach is based on the transfer function of the convolutional code which is obtained from the state diagram. The process of obtaining the transfer function and other related performance measures are described below.

2.7.1 Transfer Function of Convolutional Code

The analysis of convolutional codes is generally difficult to perform because traditional algebraic and combinatorial techniques cannot be applied. These heuristically constructed codes can be analyzed through their transfer functions. By utilizing the state diagram, the transfer function can be obtained. With the transfer function, code properties such as distance properties and the error rate performance can be easily calculated. To obtain the transfer function, the following rules are applied:

1. Break the all-zero (initial) state of the state diagram into a start state and an end state. This will be called the modified state diagram.
2. For every branch of the modified state diagram, assign the symbol D with its exponent equal to the Hamming weight of the output bits.
3. For every branch of the modified state diagram, assign the symbol J.
4. Assign the symbol N to the branch of the modified state diagram, if the branch transition is caused by an input bit 1.

For the state diagram in Figure 2.4, the modified state diagram is shown in Figure 2.14.

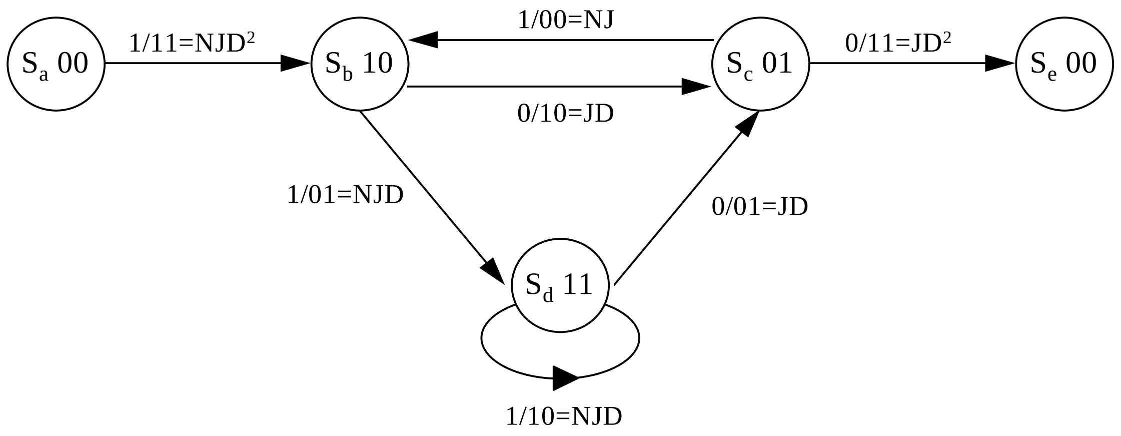

Figure 2.14: The modified state diagram of Figure 2.4 where S_a is the start state and S_e is the end state.

Nodal equations are obtained for all the states except for the start state in Figure 2.14. These results are

$$S_b = NJD^2 S_a + NJS_c$$
$$S_c = JDS_b + JDS_d$$
$$S_d = NJDS_b + NJDS_d$$
$$S_e = JD^2 S_c$$

The transfer function is defined to be

$$T(D,N,J) = \frac{S_{end}(D,N,J)}{S_{start}(D,N,J)} \tag{2.29}$$

and for Figure 2.14,

$$T(D,N,J) = \frac{S_e}{S_a}$$

By substituting and rearranging,

$$T(D,N,J) = \frac{NJ^3D^5}{1-(NJ+NJ^2)D} \qquad \text{(closed form)}$$

$$= NJ^3D^5 + (N^2J^4 + N^2J^5)D^6 + (N^3J^5 + 2N^3J^6 + N^3J^7)D^7 + \cdots$$

(expanded polynomial form)

2.7.1.1 Distance Properties

The free distance between a pair of convolutional codewords is the Hamming distance between the pair of codewords. The minimum free distance, d_{free}, is the minimum Hamming distance between all pairs of complete convolutional codewords and is defined as

$$d_{free} = \min\{d(\mathbf{y_1},\mathbf{y_2}) | \mathbf{y_1} \neq \mathbf{y_2}\} \quad \text{[Wic95]} \tag{2.30}$$

$$= \min\{w(\mathbf{y}) | \mathbf{y} \neq \mathbf{0}\} \quad \text{[Wic95]} \tag{2.31}$$

where d(•,•) is the Hamming distance between a pair of convolutional codewords and w(•) is the Hamming distance between a convolutional codeword and the all-zero codeword (the weight of the codeword). The minimum free distance corresponds to the ability of the convolutional code to estimate the best decoded bit sequence. As d_{free} increases, the performance of the convolutional code also increases. This characteristic is similar to the minimum distance for block codes. From the transfer function, the minimum free distance is identified as the lowest exponent of D. From the above transfer function for Figure 2.14, $d_{free} = 5$. Also, if N and J are set to 1, the coefficients of D^i's represent the number of paths through the trellis with weight D^i. More information about the codeword is obtained from observing the exponents of N and J. For a codeword, the exponent of N indicates the number of 1s in the input sequence, and the exponent of J indicates the length of the path that merges with the all-zero path for the first time [Pro95].

2.7.1.2 Error Probabilities

There are two error probabilities associated with convolutional codes, namely first event and bit error probabilities. The first event error probability, P_e, is the probability that an error begins at a particular time. The bit error probability, P_b, is the average number of bit errors in the decoded sequence. Usually, these error probabilities are defined using the Chernoff Bounds and are derived in [Pro95], [Rhe89], [Wic95].

For hard-decision decoding, the first event error and bit error probabilities are defined as

$$P_e < T(D, N, J)\Big|_{D=\sqrt{4p(1-p)},\, N=1,\, J=1} \tag{2.32}$$

and

$$P_b < \left.\frac{dT(D, N, J)}{dN}\right|_{D=\sqrt{4p(1-p)},\, N=1,\, J=1} \tag{2.33}$$

where

$$p = Q\left(\sqrt{\frac{2rE_b}{N_o}}\right) \tag{2.34}$$

and

$$Q(x) = \int_x^{\infty} \frac{1}{\sqrt{2\pi}} e^{-u^2/2} du \tag{2.35}$$

For soft-decision decoding, the first event error and bit error probabilities are defined as

$$P_e < T(D, N, J)\Big|_{D=e^{-rE_b/N_o},\, N=1,\, J=1} \tag{2.36}$$

and

$$P_b < \left.\frac{dT(D, N, J)}{dN}\right|_{D=e^{-rE_b/N_o},\, N=1,\, J=1} \tag{2.37}$$

Two other factors also determine the performance of the Viterbi decoder. They are commonly referred to as the decoding depth and the degree of quantization of the received signal.

2.7.2 Decoding Depth

The decoding depth is a window in time that makes a decision on the bits at the beginning of the window and accepts bits at the end of the window for metric computations. This scheme gives up the optimum ML decoding at the expense of using less memory and smaller decoding delay. It has been experimentally found that if the decoding depth is 5 times greater than the constraint length K then the error introduced by the decoding depth is negligible [Pro95].

2.7.3 Degree of Quantization

For soft-decision Viterbi decoding, the degree of the quantization on the received signal can affect the decoder performance. The performance of the Viterbi decoder improves with higher bit quantization. It has been found that an eight-level quantizer degrades the performance only slightly with respect to the infinite bit quantized case [Wic95].

2.7.4 Decoding Complexity for Convolutional Codes

For a general convolutional code, the input information sequence contains k*L bits where k is the number of parallel information bits at one time interval and L is the number of time intervals. This results in L+m stages in the trellis diagram. There are exactly 2^{k*L} distinct paths in the trellis diagram, and as a result, an exhaustive search for the ML sequence would have a computational complexity on the order of $O[2^{k*L}]$. The Viterbi algorithm reduces this complexity by performing the ML search one stage at a time in the trellis. At each node (state) of the trellis, there are 2^k calculations. The number of nodes per stage in the trellis is 2^m. Therefore, the complexity of the Viterbi algorithm is on the order of $O[(2^k)(2^m)(L+m)]$. This significantly reduces the number of calculations required to implement the ML decoding because the number of time intervals L is now a linear factor and not an exponent factor in the complexity. However, there will be an exponential increase in complexity if either k or m increases.