Convolutional codes

8

We have seen that with an (n, k) block code, codewords are constructed independently of each other. For a binary code the n bit output of an encoder depends solely on the k bits entering the encoder. The $n-k$ parity bits of a codeword depend on the k information bits that enter the encoder, each information bit affects only one codeword. Convolutional codes differ from block codes in that the encoder output is constructed not from a single input but also using some of the previous encoder inputs. Clearly memory is required to achieve this, to store inputs for further use. A convolutional code that at a given time generates n outputs from k inputs and m previous inputs is referred to as an (n, k, m) *convolutional code.*

8.1 Convolution

As their name implies, convolutional codes are based on a convolution operation and it is useful to take a look at this before considering convolutional codes. The mathematical operation of convolution can be applied to analogue functions and to discrete functions, here we address only discrete convolution. Figure 8.1 shows a shift register consisting of 4 *stages* which feed into a modulo-2 adder via the links g_1, g_2, g_3, and g_4, each stage is a 1-bit storage device. The input to the first stage also feeds into the modulo-2 adder, via the link g_0. The terms g_0, g_1, g_2, g_3, and g_4 have values 0 or 1 if the link is absent or present respectively. If all the links are present, then $g_0 = g_1 = g_2 = g_3 = g_4 = 1$. The shift register shown in Fig. 8.1 is an example of a *linear feed-forward shift register* as bits are fed towards the output and not back into the shift register.

Consider the sequence of bits

$$u = (u_0\ u_1\ u_2 \ldots)$$

entering the shift register shown in Fig. 8.1, and let

$$v = (v_0\ v_1\ v_2 \ldots)$$

be the sequence of bits leaving the modulo-2 adder as u enters. The sequences u and v are referred to as the *input sequence* and the *output sequence* respectively. If we now

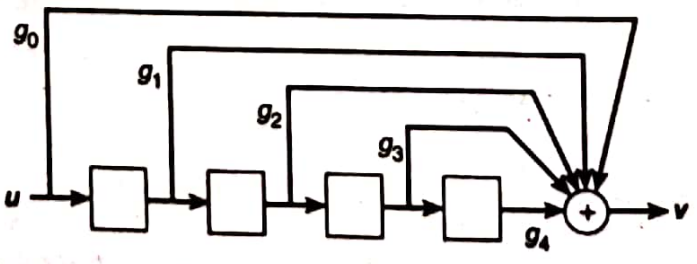


Fig. 8.1 Linear-feedforward shift register with 4 stages.

assume that not all the links in Fig. 8.1 are necessarily present, so that g_0, g_1, g_2, g_3, and g_4 are 0 or 1, then once u_0 enters the register, the output from the modulo-2 adder will be $v_0 = u_0 g_0$ since all the other inputs to the adder are zero. If $u_0 = 0$, or if the link g_0 is absent, so that $g_0 = 0$, then $v_0 = 0$, otherwise $v_0 = 1$. When u_1 enters the register, the bit u_0 moves into the second stage. If $g_1 = 1$, then u_0 is fed into the adder and likewise, if $g_0 = 1$, then u_0 is also fed into the adder. The output from the adder is therefore $v_1 = u_1 g_0 + u_0 g_1$. At the next input the adder output is $v_2 = u_2 g_0 + u_1 g_1 + u_0 g_2$ followed by $v_3 = u_3 g_0 + u_2 g_1 + u_1 g_2 + u_0 g_3$ and $v_4 = u_4 g_0 + u_3 g_1 + u_2 g_2 + u_1 g_3 + u_0 g_4$. In summary the register output is:

$$\begin{aligned} v_0 &= u_0 g_0 \\ v_1 &= u_1 g_0 + u_0 g_1 \\ v_2 &= u_2 g_0 + u_1 g_1 + u_0 g_2 \\ v_3 &= u_3 g_0 + u_2 g_1 + u_1 g_2 + u_0 g_3 \\ v_4 &= u_4 g_0 + u_3 g_1 + u_2 g_2 + u_1 g_3 + u_0 g_4. \end{aligned} \tag{8.1}$$

Note that the first bit u_0 to enter the register has affected all 5 outputs. As the next bit u_5 enters the shift register u_0 will cease to contribute to the output sequence. In a register with m stages each input contributes towards $m+1$ outputs.

Example 8.1

Consider the shift register shown in Fig. 8.1 and let $g_0 = 1$, $g_1 = 1$, $g_2 = 0$, $g_3 = 0$ and $g_4 = 1$. Determine the output sequence for the input sequence $\boldsymbol{u} = (1\ 1\ 0\ 1\ 0\ 1\ \ldots)$.

The first 5 inputs are $u_0 = 1$, $u_1 = 1$, $u_2 = 0$, $u_3 = 1$, and $u_4 = 0$. Using eqns 8.1 gives

$$\begin{aligned} v_0 &= 1 \cdot 1 = 1 \\ v_1 &= 1 \cdot 1 + 1 \cdot 1 = 0 \\ v_2 &= 0 \cdot 1 + 1 \cdot 1 + 1 \cdot 0 = 1 \\ v_3 &= 1 \cdot 1 + 0 \cdot 1 + 1 \cdot 0 + 1 \cdot 0 = 1 \\ v_4 &= 0 \cdot 1 + 1 \cdot 1 + 0 \cdot 0 + 1 \cdot 0 + 1 \cdot 1 = 0 \end{aligned}$$

as the first 5 bits of the output sequence. The 6th bit is given by

$$\begin{aligned} v_5 &= u_5 g_0 + u_4 g_1 + u_3 g_2 + u_2 g_3 + u_1 g_4 \\ &= 1 \cdot 1 + 0 \cdot 1 + 1 \cdot 0 + 0 \cdot 0 + 1 \cdot 1 \\ &= 0. \end{aligned}$$

The output sequence is therefore $(1\ 0\ 1\ 1\ 0\ 0\ \ldots)$. □

For a finite input sequence, we need to ensure that the last input feeds through the register and appears at the output. To achieve this an additional m zero inputs are required. Consider Fig. 8.1 again, with the input sequence $\boldsymbol{u} = (u_0\, u_1\, u_2\, u_3\, u_4\, u_5\, u_6\, u_7)$. When u_7 enters the register the output is

$$v_7 = u_7 g_0 + u_6 g_1 + u_5 g_2 + u_4 g_3 + u_3 g_4.$$

Assuming now an additional 4 inputs u_8, u_9, u_{10}, and u_{11} then there are a further 4 outputs

$$\begin{aligned} v_8 &= u_8g_0 + u_7g_1 + u_6g_2 + u_5g_3 + u_4g_4 \\ v_9 &= u_9g_0 + u_8g_1 + u_7g_2 + u_6g_3 + u_5g_4 \\ v_{10} &= u_{10}g_0 + u_9g_1 + u_8g_2 + u_7g_3 + u_6g_4 \\ v_{11} &= u_{11}g_0 + u_{10}g_1 + u_9g_2 + u_8g_3 + u_7g_4 \end{aligned}$$

and setting $u_8 = u_9 = u_{10} = u_{11} = 0$ gives

$$\begin{aligned} v_8 &= u_7g_1 + u_6g_2 + u_5g_3 + u_4g_4 \\ v_9 &= u_7g_2 + u_6g_3 + u_5g_4 \\ v_{10} &= u_7g_3 + u_6g_4 \\ v_{11} &= u_7g_4. \end{aligned}$$

All the 7 inputs have now left the shift register and the output sequence is completed. Any further zero inputs will give a zero output. By feeding in the additional m 0 bits the register is said to be cleared or returned to its *zero state*. The m 0 bits can be appended to the input sequence $\boldsymbol{u}$ but should not be confused with the information bits.

Example 8.2

Figure 8.2 shows a shift register with 3 stages and $g_0 = 1$, $g_1 = 0$, $g_2 = 1$, and $g_3 = 1$. Determine the output sequence given the input sequence $\boldsymbol{u} = (1\ 0\ 0\ 1)$.

As $\boldsymbol{u} = (u_0\, u_1\, u_2\, u_3)$ enters the shift register, the output from the adder is

$$\begin{aligned} v_0 &= u_0g_0 \\ v_1 &= u_1g_0 + u_0g_1 \\ v_2 &= u_2g_0 + u_1g_1 + u_0g_2 \\ v_3 &= u_3g_0 + u_2g_1 + u_1g_2 + u_0g_3. \end{aligned}$$

Substituting the values of g_0, g_1, g_2, and g_3 gives

$$\begin{aligned} v_0 &= u_0 \\ v_1 &= u_1 \\ v_2 &= u_2 + u_0 \\ v_3 &= u_3 + u_1 + u_0 \end{aligned}$$

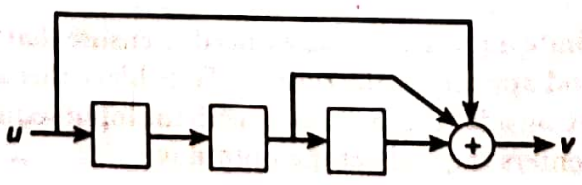


Fig. 8.2 Register with 3 stages.

and for $u_0 = 1$, $u_1 = 0$, $u_2 = 0$, and $u_3 = 1$, we get $v_0 = 1$, $v_1 = 0$, $v_2 = 1$, and $v_3 = 0$. To return the register to its zero state we need another 3 inputs, $u_4 = u_5 = u_6 = 0$. This gives

$$v_4 = u_4g_0 + u_3g_1 + u_2g_2 + u_1g_3 = 0$$
$$v_5 = u_5g_0 + u_4g_1 + u_3g_2 + u_2g_3 = 1$$
$$v_6 = u_6g_0 + u_5g_1 + u_4g_2 + u_3g_3 = 1.$$

The output sequence is therefore $v = (v_0\, v_1\, v_2\, v_3\, v_4\, v_5\, v_6) = (1\ 0\ 1\ 0\ 0\ 1\ 1)$. □

Returning now to eqns 8.1, for the shift register shown in Fig. 8.1, given the input u_j the output from the modulo-2 adder is

$$v_j = u_jg_0 + u_{j-1}g_1 + u_{j-2}g_2 + u_{j-3}g_3 + u_{j-4}g_4$$

or more concisely

$$v_j = \sum_{r=0}^{4} u_{j-r}g_r$$

where $u_{j-r} = 0$ if $j < r$ (this defines $u_{-1} = u_{-2} = u_{-3} = u_{-4} = 0$). For a shift register with m stages and links $g_0, g_1, g_2, \ldots, g_m$ feeding into a modulo-2 adder, the output from the adder is given by

$$v_j = \sum_{r=0}^{m} u_{j-r}g_r \tag{8.2}$$

where again $u_{j-r} = 0$ if $j < r$. Equation 8.2 shows that each output v_j is a *convolution* of $(m+1)$ inputs with $g_0, g_1, \ldots, g_m$. We can think of convolution as a process in which a *window* of $(m+1)$ bits are multiplied by $g_0, g_1, \ldots, g_m$ and added together. The window is then slid along the input sequence by 1 bit and multiplication and summation repeated. The process continues until the window has covered the entire input sequence. Equation 8.2 can be expressed in terms of the input and output sequences as

$$v = u * g \tag{8.3}$$

where the operation $*$ denotes convolution and $g = (g_0\, g_1 \ldots g_m)$ is referred to as a *generator sequence*. The generator sequence can be obtained by inspection of the shift register or alternatively by considering the *impulse response* when the input sequence is a 1 followed by m 0s, i.e. $u = (1\ 0\ 0\ 0 \ldots 0)$. For example, consider again Fig. 8.1 this time with input sequence $u = (u_0\, u_1\, u_2\, u_3\, u_4)$ where $u_0 = 1$ and $u_1 = u_2 = u_3 = u_4 = 0$. Substituting u_0, u_1, u_2, u_3, and u_4 into eqns 8.1 gives

$$v_0 = 1g_0 = g_0$$
$$v_1 = 0g_0 + 1g_1 = g_1$$
$$v_2 = 0g_0 + 0g_1 + 1g_2 = g_2$$
$$v_3 = 0g_0 + 0g_1 + 0g_2 + 1g_3 = g_3$$
$$v_4 = 0g_0 + 0g_1 + 0g_2 + 0g_3 + 1g_4 = g_4.$$

Hence the output sequence is

$$v = (v_0\, v_1\, v_2\, v_3\, v_4) = (g_0\, g_1\, g_2\, g_3\, g_4) = g$$

and therefore the impulse response gives the generator sequence g. The generator sequence g should not be interpreted as the generator sequence of the shift register but rather of the output from the modulo-2 adder. A line of stages feeding into more than one modulo-2 adder will have a generator sequence for each adder. Example 8.3 below illustrates this.

Example 8.3

Figure 8.3 shows a shift register with 3 stages feeding into two modulo-2 adders. The output v_1 does not have an input directly from u nor from the first stage, but has inputs from the second and third stages. Hence the generator sequence of v_1 is $g_1 = (0\ 0\ 1\ 1)$. Likewise we can see that the generator sequence for v_2 is $g_2 = (1\ 0\ 1\ 1)$. Using eqn 8.2 gives

$$\begin{aligned}
v_0 &= u_0g_0 + u_{-1}g_1 + u_{-2}g_2 + u_{-3}g_3\\
v_1 &= u_1g_0 + u_0g_1 + u_{-1}g_2 + u_{-2}g_3\\
v_2 &= u_2g_0 + u_1g_1 + u_0g_2 + u_{-1}g_3\\
v_3 &= u_3g_0 + u_2g_1 + u_1g_2 + u_0g_3\\
v_4 &= u_4g_0 + u_3g_1 + u_2g_2 + u_1g_3
\end{aligned}$$

for an input $u = (u_0\, u_1\, u_2\, u_3\, u_4)$ and substituting the values of g_1 and g_2 we get

$$\begin{aligned}
v_0 &= 0\\
v_1 &= 0\\
v_2 &= u_0\\
v_3 &= u_1 + u_0\\
v_4 &= u_2 + u_1
\end{aligned}$$

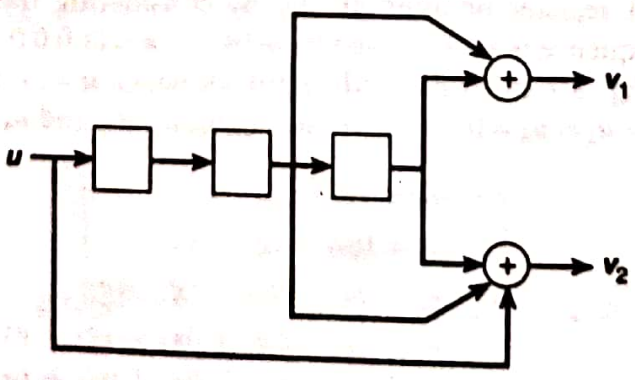


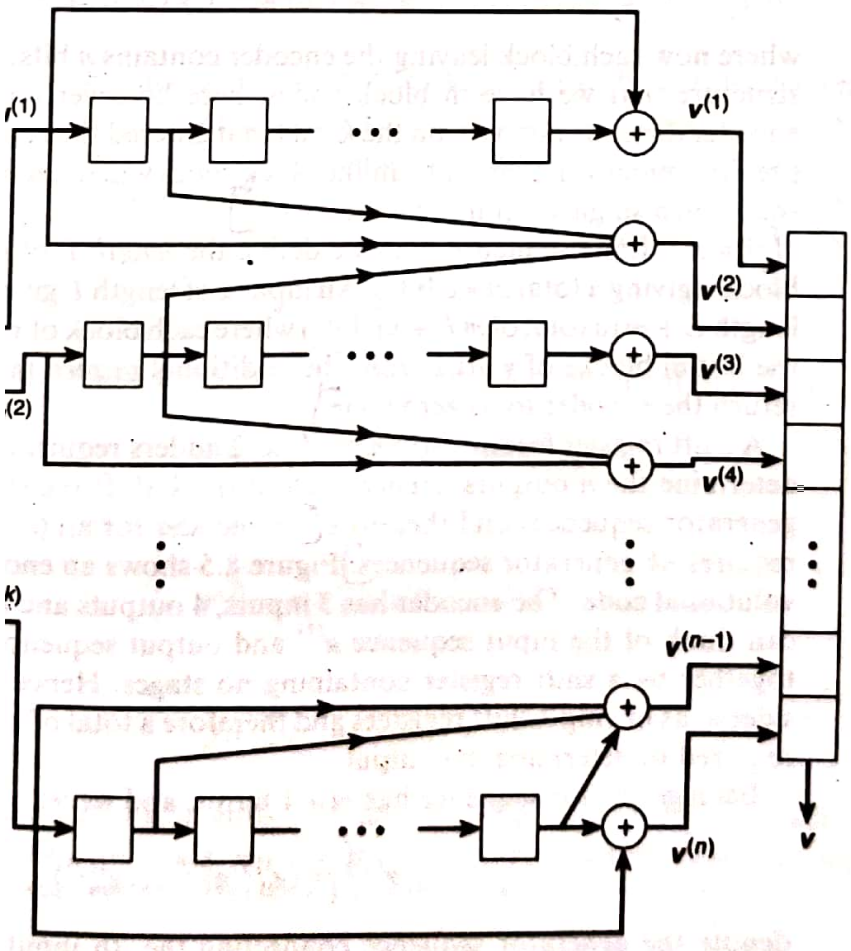
Fig. 8.3 Register with 2 outputs.

$$v_0 = u_0$$
$$v_1 = u_1$$
$$v_2 = u_2 + u_0$$
$$v_3 = u_3 + u_1 + u_0$$
$$v_4 = u_4 + u_2 + u_1$$

ıents of v_1 and v_2 respectively. □

nvolutional codes

ion of convolution now established, we move on to looking at encod-
ional codes. Figure 8.4 shows an encoder for an arbitrary (n, k, m)
l code. The encoder consists of a bank of k linear-feedforward shift
ch do not necessarily have the same number of stages. For an (n, k, m)
l code, the maximum number of stages m in any shift register is known
y order of the code. The encoder output is taken from n modulo-2



r for an (n, k, m) convolutional code.

adders whose inputs are taken from various stages (depending upon the particular code). Each bit in the encoder can affect any of the n outputs, and given that a bit can stay in the encoder for up to $m+1$ inputs then each bit can affect up to $n(m+1)$ output bits. This measure of the extent to which an input bit affects the output is referred to as the *constraint length*.

The input sequence to the ith shift register is denoted by

$$u^{(i)} = (u_0^{(i)} u_1^{(i)} u_2^{(i)} \ldots) \tag{8.4}$$

and bits are fed into their respective shift register one bit at a time. Viewing the encoder on the whole, bits enter in blocks of k bits at a time, and we can express the input sequence to the encoder as

$$u = (u_0^{(1)} u_0^{(2)} \ldots u_0^{(k)}, u_1^{(1)} u_1^{(2)} \ldots u_1^{(k)}, \ldots). \tag{8.5}$$

Here the first input to the encoder is the block of k bits $u_0^{(1)} u_0^{(2)} \ldots u_0^{(k)}$ followed by the block $u_1^{(1)} u_1^{(2)} \ldots u_1^{(k)}$ and so forth. The output sequence of the jth modulo-2 adder is

$$v^{(j)} = (v_0^{(j)} v_1^{(j)} v_2^{(j)} \ldots) \tag{8.6}$$

and again viewing the encoder on the whole the output sequence is

$$v = (v_0^{(1)} v_0^{(2)} \ldots v_0^{(n)}, v_1^{(1)} v_1^{(2)} \ldots v_1^{(n)}, \ldots) \tag{8.7}$$

where now each block leaving the encoder contains n bits. This may sound like the structure that we have in block codes, here however an n-bit block leaving the encoder depends not only on the k-bits that entered the encoder but also on up to m previous blocks. This is quite unlike block codes where each n-bit codeword depends solely on a single k-bit information word.

If u is a finite sequence then we define the *length* L of u as the number of k-bit blocks (giving a total of kL bits). An input u of length L gives an output sequence v of length $L+m$ (a total of $n(L+m)$ bits) where each block of v contains n bits and where the last m blocks of v arise from the additional m zero inputs that are required to return the encoder to its zero state.

A shift register feeding into n modulo-2 adders requires n generator sequences to determine the n outputs. Hence each of the k shift registers in Fig. 8.4 requires n generator sequences and therefore the encoder for an (n, k, m) convolutional code requires nk generator sequences. Figure 8.5 shows an encoder for the (4, 3, 2) convolutional code. The encoder has 3 inputs, 4 outputs and memory order $m=2$. We can think of the input sequence $u^{(1)}$ and output sequence $v^{(1)}$ as being connected together by a shift register containing no stages. Hence the encoder can be considered as having 3 shift registers and therefore a total of 12 generator sequences are required to determine the output.

Each generator sequence has $m+1$ terms, and we let

$$g^{(i,j)} = (g_0^{(i,j)} g_1^{(i,j)} \ldots g_m^{(i,j)})$$

denote the generator sequence connecting the ith input to the jth output. Then by considering the impulse response of each shift register or by inspecting the

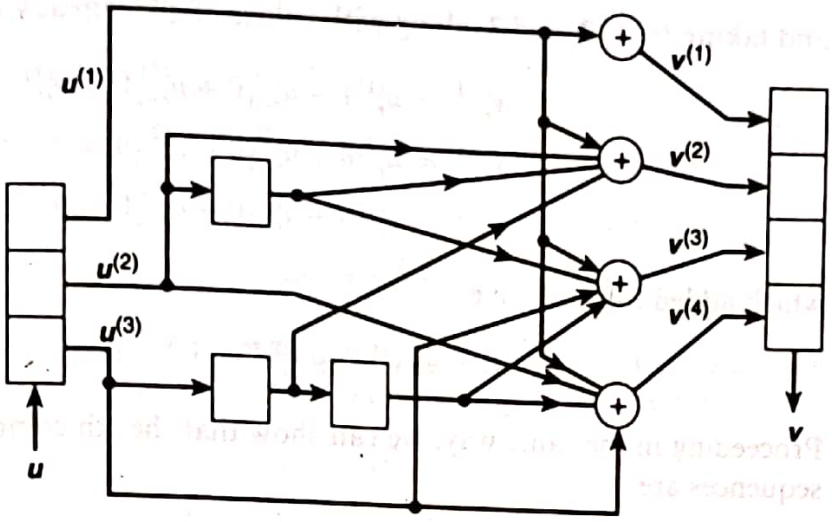


Fig. 8.5 Encoder for the (4, 3, 2) convolutional code.

connections to each adder we find that

$$\begin{aligned} g^{(1,1)} &= (1\ 0\ 0),\ g^{(1,2)} = (1\ 0\ 0),\ g^{(1,3)} = (1\ 0\ 0),\ g^{(1,4)} = (1\ 0\ 0) \\ g^{(2,1)} &= (0\ 0\ 0),\ g^{(2,2)} = (1\ 1\ 0),\ g^{(2,3)} = (0\ 1\ 0),\ g^{(2,4)} = (1\ 0\ 0) \\ g^{(3,1)} &= (0\ 0\ 0),\ g^{(3,2)} = (0\ 1\ 0),\ g^{(3,3)} = (1\ 0\ 1),\ g^{(3,4)} = (1\ 0\ 1). \end{aligned} \tag{8.8}$$

In Section 8.1 we saw that given a generator sequence g, the output sequence v for an input sequence u is $v = u * g$ (eqn 8.3). For the encoder shown in Fig. 8.5, each output sequence has contributions from up to 3 inputs and therefore eqn 8.3 has to be modified to take the additional inputs into account. This is achieved by adding the term $u^{(i)} * g^{(i,j)}$ term to $v^{(j)}$ for each input $u^{(i)}$. For convenience we define

$$v^{(i,j)} = u^{(i)} * g^{(i,j)} \tag{8.9}$$

as the jth output arising from the ith input. The rth component of $v^{(i,j)}$ is

$$v_r^{(i,j)} = u_r^{(i)} g_0^{(i,j)} + u_{r-1}^{(i)} g_1^{(i,j)} + u_{r-2}^{(i)} g_2^{(i,j)}. \tag{8.10}$$

The jth output sequence can be expressed as

$$v^{(j)} = v^{(1,j)} + v^{(2,j)} + v^{(3,j)} \tag{8.11}$$

which has

$$v_r^{(j)} = v_r^{(1,j)} + v_r^{(2,j)} + v_r^{(3,j)} \tag{8.12}$$

as its rth component. From eqns 8.10 and 8.12 the components of each output sequence can be determined. Consider first the components of the output sequence $v^{(1)}$. The rth component of $v^{(1)}$ arising from the ith input is

$$v_r^{(i,1)} = u_r^{(i)} g_0^{(i,1)} + u_{r-1}^{(i)} g_1^{(i,1)} + u_{r-2}^{(i)} g_2^{(i,1)}$$

and taking $i = 1$, 2, and 3, along with values of $g^{(i,j)}$ already given, we obtain

$$v_r^{(1,1)} = u_r^{(1)}1 + u_{r-1}^{(1)}0 + u_{r-2}^{(1)}0 = u_r^{(1)}$$
$$v_r^{(2,1)} = u_r^{(2)}0 + u_{r-1}^{(2)}0 + u_{r-2}^{(2)}0 = 0$$
$$v_r^{(3,1)} = u_r^{(3)}0 + u_{r-1}^{(3)}0 + u_{r-2}^{(3)}0 = 0$$

which added together give

$$v_r^{(1)} = v_r^{(1,1)} + v_r^{(2,1)} + v_r^{(3,1)} = u_r^{(1)}.$$

Proceeding in the same way, we can show that the rth component of all 4 output sequences are

$$\begin{aligned} v_r^{(1)} &= u_r^{(1)} \\ v_r^{(2)} &= u_r^{(1)} + u_r^{(2)} + u_{r-1}^{(2)} + u_{r-1}^{(3)} \\ v_r^{(3)} &= u_r^{(1)} + u_{r-1}^{(2)} + u_r^{(3)} + u_{r-2}^{(3)} \\ v_r^{(4)} &= u_r^{(1)} + u_r^{(2)} + u_r^{(3)} + u_{r-2}^{(3)} \end{aligned} \tag{8.13}$$

and the output sequence leaving the encoder is

$$v = (v_0^{(1)}v_0^{(2)}v_0^{(3)}v_0^{(4)}, v_1^{(1)}v_1^{(2)}v_1^{(3)}v_1^{(4)}, v_2^{(1)}v_2^{(2)}v_2^{(3)}v_2^{(4)} \ldots).$$

Example 8.4

Determine the output sequence from the (4, 3, 2) convolutional encoder, shown in Fig. 8.5, given the input sequences $u^{(1)} = (1\ 0\ 1)$, $u^{(2)} = (1\ 1\ 0)$, and $u^{(3)} = (0\ 1\ 1)$.

The input sequences are of the form $u = (u_0\, u_1\, u_2)$ with $u_r = 0$ if $r < 0$. We start by taking $r = 0$, then from eqns 8.13

$$\begin{aligned} v_0^{(1)} &= u_0^{(1)} \\ v_0^{(2)} &= u_0^{(1)} + u_0^{(2)} \\ v_0^{(3)} &= u_0^{(1)} + u_0^{(3)} \\ v_0^{(4)} &= u_0^{(1)} + u_0^{(2)} + u_0^{(3)} \end{aligned}$$

and substituting $u_0^{(1)} = 1$, $u_0^{(2)} = 1$, $u_0^{(3)} = 0$ gives $v_0^{(1)} = 1$, $v_0^{(2)} = 0$, $v_0^{(3)} = 1$, and $v_0^{(4)} = 0$.

Next take $r = 1$, this gives

$$\begin{aligned} v_1^{(1)} &= u_1^{(1)} = 0 \\ v_1^{(2)} &= u_1^{(1)} + u_1^{(2)} + u_0^{(2)} + u_0^{(3)} = 0 \\ v_1^{(3)} &= u_1^{(1)} + u_0^{(2)} + u_1^{(3)} = 0 \\ v_1^{(4)} &= u_1^{(1)} + u_1^{(2)} + u_1^{(3)} = 0 \end{aligned}$$

and for $r=2$

$$v_2^{(1)} = u_2^{(1)} = 1$$
$$v_2^{(2)} = u_2^{(1)} + u_2^{(2)} + u_1^{(2)} + u_1^{(3)} = 1$$
$$v_2^{(3)} = u_2^{(1)} + u_1^{(2)} + u_2^{(3)} + u_0^{(3)} = 1$$
$$v_2^{(4)} = u_2^{(1)} + u_2^{(2)} + u_2^{(3)} + u_0^{(3)} = 0.$$

Because the inputs are finite sequences and $m=2$ we need to consider 2 more 0 inputs to return the encoder back to the zero state. This requires

$$u_3^{(1)} = u_3^{(2)} = u_3^{(3)} = 0$$
$$u_4^{(1)} = u_4^{(2)} = u_4^{(3)} = 0.$$

For $r=3$ we get

$$v_3^{(1)} = u_3^{(1)} = 0$$
$$v_3^{(2)} = u_3^{(1)} + u_3^{(2)} + u_2^{(2)} + u_2^{(3)} = 1$$
$$v_3^{(3)} = u_3^{(1)} + u_2^{(2)} + u_3^{(3)} + u_1^{(3)} = 1$$
$$v_3^{(4)} = u_3^{(1)} + u_3^{(2)} + u_3^{(3)} + u_1^{(3)} = 1$$

and $r=4$ gives

$$v_4^{(1)} = u_4^{(1)} = 0$$
$$v_4^{(2)} = u_4^{(1)} + u_4^{(2)} + u_3^{(2)} + u_3^{(3)} = 0$$
$$v_4^{(3)} = u_4^{(1)} + u_3^{(2)} + u_4^{(3)} + u_2^{(3)} = 1$$
$$v_4^{(4)} = u_4^{(1)} + u_4^{(2)} + u_4^{(3)} + u_2^{(3)} = 1.$$

Therefore the output sequence is

$$v = (1\,0\,1\,0,\ 0\,0\,0\,0,\ 1\,1\,1\,0,\ 0\,1\,1\,1,\ 0\,0\,1\,1). \qquad \square$$

The rth component of the output sequence, given by eqns 8.13, can be obtained directly from the encoder shown in Fig. 8.5. Let's assume that the input sequence is at the rth component, so that $u_r^{(1)}$, $u_r^{(2)}$, and $u_r^{(3)}$ are the inputs, then the 4 outputs can be determined by inspecting the encoder. The output $v^{(1)}$ has only one input to its modulo-2 adder, namely that which arrives directly from $u^{(1)}$ and so $v_r^{(1)} = u_r^{(1)}$, as given by eqns 8.13. The output $v^{(2)}$ has:

(1) inputs directly from $u^{(1)}$, so contributing $u_r^{(1)}$ to $v^{(2)}$;
(2) inputs directly from $u^{(2)}$, which contributes $u_r^{(2)}$;
(3) a contribution from $u^{(2)}$ that is delayed by 1 stage and therefore contributes $u_{r-1}^{(2)}$;
(4) a contribution from $u^{(3)}$ that is also delayed by 1 stage and therefore contributes $u_{r-1}^{(3)}$ to $v^{(2)}$.

Adding together the four contributions gives $u_r^{(1)} + u_r^{(2)} + u_{r-1}^{(2)} + u_{r-1}^{(3)}$ as the rth component of $v^{(2)}$, which again is in agreement with that given by eqns 8.13. Likewise we can verify that $v_r^{(3)}$ and $v_r^{(4)}$ are as given by eqns 8.13.

As with blockcodes the error-control properties of convolutional codes depend on the distance characteristics of the resulting encoded sequences. However with convolutional codes there are several minimum distance measures, the most important measure being the minimum *free distance* defined as the minimum distance between any two encoded sequences. If two sequences $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$ are of different length then zeros are added to the shorter corresponding input sequence $\boldsymbol{u}_1$ or $\boldsymbol{u}_2$ so that the sequences are the same length. Hence the definition of free distance includes all sequences and not just sequences with the same length. Convolutional codes are linear codes and therefore the sum of two encoded sequences $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$ gives another encoded sequence $\boldsymbol{v}_3$ where the weight of $\boldsymbol{v}_3$ equals the distance between $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$, and $\boldsymbol{v}_3 \neq 0$ if $\boldsymbol{v}_1 \neq \boldsymbol{v}_2$. Hence the free distance of a convolutional code is given by the minimum-weight sequence of any length produced by a nonzero input sequence.

8.3 Generator matrices for convolutional codes

The use of convolution to derive the output sequences is rather tedious and as might be expected, the use of matrices can help to simplify encoding. We have seen that the output sequence $\boldsymbol{v} = \boldsymbol{u} * \boldsymbol{g}$, where $\boldsymbol{u}$ is an input sequence, $\boldsymbol{g}$ is a generator sequence and $*$ is the convolution operation. This equation can be expressed in matrix form, and the convolution operation replaced by matrix multiplication, by defining a *generator matrix* G such that

$$\boldsymbol{v} = \boldsymbol{u}\boldsymbol{G}. \tag{8.14}$$

The matrix $\boldsymbol{G}$ is constructed from the components of the generator sequences and for an (n, k, m) convolutional code

$$\boldsymbol{G} = \begin{bmatrix} \boldsymbol{G}_0 & \boldsymbol{G}_1 & \boldsymbol{G}_2 & \dots & \boldsymbol{G}_m & \boldsymbol{0} & \boldsymbol{0} & \dots \\ \boldsymbol{0} & \boldsymbol{G}_0 & \boldsymbol{G}_1 & \boldsymbol{G}_2 & \dots & \boldsymbol{G}_m & \boldsymbol{0} & \dots \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{G}_0 & \boldsymbol{G}_1 & \boldsymbol{G}_2 & \dots & \boldsymbol{G}_m & \dots \\ \cdot & & & & & \dots & & \\ \cdot & & & & & & \dots & \\ \cdot & & & & & & & \dots \end{bmatrix} \tag{8.15}$$

where $\boldsymbol{G}_r$ is the k by n matrix

$$\boldsymbol{G}_r = \begin{bmatrix} g_r^{(1,1)} & g_r^{(1,2)} & \dots & g_r^{(1,n)} \\ g_r^{(2,1)} & g_r^{(2,2)} & \dots & g_r^{(2,n)} \\ \vdots & & & \vdots \\ g_r^{(k,1)} & g_r^{(k,2)} & \dots & g_r^{(k,n)} \end{bmatrix} \tag{8.16}$$

and **0** is a k by n zero matrix. Each row in $\boldsymbol{G}$ is obtained from the previous row by shifting all the matrices one place to the right. If $\boldsymbol{u}$ has finite length L then $\boldsymbol{G}$ has L rows and $L+m$ columns. On substituting eqn 8.16 into 8.15 the matrix $\boldsymbol{G}$ has kL rows and $n(L+m)$ columns. For the (4, 3, 2) code already considered, we have

$$\boldsymbol{G}_r = \begin{bmatrix} g_r^{(1,1)} & g_r^{(1,2)} & g_r^{(1,3)} & g_r^{(1,4)} \\ g_r^{(2,1)} & g_r^{(2,2)} & g_r^{(2,3)} & g_r^{(2,4)} \\ g_r^{(3,1)} & g_r^{(3,2)} & g_r^{(3,3)} & g_r^{(3,4)} \end{bmatrix}.$$

Using the generator sequences already given for the (4, 3, 2) code (see section 8.2) we get:

$$\boldsymbol{G}_0 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix}$$

$$\boldsymbol{G}_1 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

$$\boldsymbol{G}_2 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix}.$$

The generator matrix for the (4, 3, 2) convolutional code is therefore

$$\boldsymbol{G} = \begin{bmatrix} 1\,1\,1\,1 & 0\,0\,0\,0 & 0\,0\,0\,0 & \mathbf{0} & \mathbf{0}\ \ldots \\ 0\,1\,0\,1 & 0\,1\,1\,0 & 0\,0\,0\,0 & & \\ 0\,0\,1\,1 & 0\,1\,0\,0 & 0\,0\,1\,1 & & \\ \mathbf{0} & 1\,1\,1\,1 & 0\,0\,0\,0 & 0\,0\,0\,0 & \mathbf{0}\ \ldots \\ & 0\,1\,0\,1 & 0\,1\,1\,0 & 0\,0\,0\,0 & \\ & 0\,0\,1\,1 & 0\,1\,0\,0 & 0\,0\,1\,1 & \\ \mathbf{0} & \mathbf{0} & 1\,1\,1\,1 & 0\,0\,0\,0 & 0\,0\,0\,0 \\ & & 0\,1\,0\,1 & 0\,1\,1\,0 & 0\,0\,0\,0 \\ & & 0\,0\,1\,1 & 0\,1\,0\,0 & 0\,0\,1\,1 \\ & & & \ldots & \\ & & & \ldots & \\ & & & & \ldots \end{bmatrix} \qquad (8.17)$$

where **0** is a 3 by 4 zero matrix. Once the generator matrix is established finding the output sequence for a given input sequence is straightforward. Reconsider Example 8.4 in which the output sequence for the input sequences $\boldsymbol{u}^{(1)} = (1\ 0\ 1)$, $\boldsymbol{u}^{(2)} = (1\ 1\ 0)$ and $\boldsymbol{u}^{(3)} = (0\ 1\ 1)$ was found using convolution. The input sequence to the register is

$$\begin{aligned} \boldsymbol{u} &= (u_0^{(1)} u_0^{(2)} u_0^{(3)}, u_1^{(1)} u_1^{(2)} u_1^{(3)}, u_2^{(1)} u_2^{(2)} u_2^{(3)}) \\ &= (1\,1\,0, 0\,1\,1, 1\,0\,1). \end{aligned}$$

The generator matrix G is the 9 by 20 matrix

$$G = \begin{bmatrix} 1111 & 0000 & 0000 & 0000 & 0000 \\ 0101 & 0110 & 0000 & 0000 & 0000 \\ 0011 & 0100 & 0011 & 0000 & 0000 \\ 0000 & 1111 & 0000 & 0000 & 0000 \\ 0000 & 0101 & 0110 & 0000 & 0000 \\ 0000 & 0011 & 0100 & 0011 & 0000 \\ 0000 & 0000 & 1111 & 0000 & 0000 \\ 0000 & 0000 & 0101 & 0110 & 0000 \\ 0000 & 0000 & 0011 & 0100 & 0011 \end{bmatrix}$$

and using $v = uG$ gives

$$v = (1010, 0000, 1110, 0111, 0011)$$

which is in agreement with the answer obtained in Example 8.4.

8.4 Generator polynomials for convolutional codes

Whilst the use of generator matrices is an improvement over the use of convolution, it is still nevertheless awkward due to the large size of the generator matrices. However, encoding can be further simplified through the use of *generator polynomials*. Consider the input sequence $u = (u_0 u_1 u_2 \dots)$, it can be represented by the polynomial

$$u(D) = u_0 + u_1 D + u_2 D^2 + \cdots \tag{8.18}$$

where D is the *unit-delay operator* and represents a delay of 1 bit, D^2 represents a 2-bit delay and so forth. In eqn 8.18 u_1 is interpreted as being delay by 1 bit relative to u_0, that is, it arrives 1 bit later than u_0. The bit u_2 is delayed by 2 bits relative to u_0 and 1 bit relative to u_1. Given an output sequence $v = (v_0 v_1 v_2 \dots)$ we can likewise write

$$v(D) = v_0 + v_1 D + v_2 D^2 + \cdots \tag{8.19}$$

The polynomials $u(D)$ and $v(D)$ are referred to as the *input polynomial* and *output polynomial* respectively. If $v(D)$ is the output arising from $u(D)$ then

$$v(D) = u(D)g(D) \tag{8.20}$$

where $g(D)$ is a *generator polynomial*. The product $u(D)g(D)$ is constructed using polynomial multiplication subject to modulo-2 arithmetic.

Example 8.5
Given the arbitrary polynomials $u(D) = 1 + D^2 + D^3$ and $g(D) = 1 + D^3$ then

$$\begin{aligned} v(D) &= (1 + D^2 + D^3)(1 + D^3) \\ &= 1 + D^3 + D^2 + D^5 + D^3 + D^6 \\ &= 1 + D^2 + D^5 + D^6. \end{aligned}$$

□

The generator polynomial can be determined directly from the encoder, in the same way as the generator sequences are determined. Each link to a modulo-2 adder contributes a D^r term to the generator polynomial where r is the number of stages that a bit has to pass through to arrive at the adder. The rth component of the generator sequence $\boldsymbol{g} = (g_0 g_1 g_2 \cdots g_m)$ is 0 or 1 depending on whether, after the rth stage there is a link to the adder, and therefore the rth component in the generator polynomial can be written as $g_r D^r$. Hence, for m stages, the generator polynomial can be expressed as

$$g(D) = g_0 + g_1 D + g_2 D^2 + \cdots + g_m D^m. \tag{8.21}$$

Example 8.6

Determine the generator polynomial of the shift register shown in Fig. 8.2. Hence find the output sequence given the input sequence $\boldsymbol{u} = (1\ 0\ 0\ 1)$.

We have already seen that the generator sequence of the encoder shown in Fig. 8.2 is $\boldsymbol{g} = (1\ 0\ 1\ 1)$ and the generator polynomial is therefore

$$g(D) = g_0 + g_1 D + g_2 D^2 + g_3 D^3 = 1 + D^2 + D^3.$$

The input polynomial corresponding to the input sequence $\boldsymbol{u} = (1\ 0\ 0\ 1)$ is

$$u(D) = u_0 + u_1 D + u_2 D^2 + u_3 D^3 = 1 + D^3$$

and the output polynomial is therefore

$$\begin{aligned} v(D) &= u(D)g(D) \\ &= (1 + D^3)(1 + D^2 + D^3) \\ &= 1 + D^2 + D^5 + D^6 \end{aligned}$$

giving the output sequence

$$\boldsymbol{v} = (v_0\ v_1\ v_2\ v_3\ v_4\ v_5\ v_6) = (1\ 0\ 1\ 0\ 0\ 1\ 1).$$

The results agree with that obtained using convolution in Example 8.2. □

An (n, k, m) convolutional code has k generator polynomials for each of the n output sequences and therefore a total of nk generator polynomials. Let

$$g^{(i,j)}(D) = \sum_{r=0}^{m} g_r^{(i,j)} D^r \tag{8.22}$$

be the generator polynomial for the jth output arising from the ith input, where the polynomial coefficients $g_r^{(i,j)}$ are the components of the generator sequence $\boldsymbol{g}^{(i,j)}$. The (4, 3, 2) encoder shown in Fig. 8.5 has 12 generator polynomials of the form

$$g^{(i,j)}(D) = g_0^{(i,j)} + g_1^{(i,j)} D + g_2^{(i,j)} D^2.$$

Taking $i = 1, 2$, and 3, and $j = 1, 2, 3$, and 4, along with the generator sequences given in Section 8.2, gives

$$g^{(1,1)}(D) = 1,\ g^{(1,2)}(D) = 1,\ g^{(1,3)}(D) = 1,\ g^{(1,4)}(D) = 1$$
$$g^{(2,1)}(D) = 0,\ g^{(2,2)}(D) = 1 + D,\ g^{(2,3)}(D) = D,\ g^{(2,4)}(D) = 1$$
$$g^{(3,1)}(D) = 0,\ g^{(3,2)}(D) = D,\ g^{(3,3)}(D) = 1 + D^2,\ g^{(3,4)}(D) = 1 + D^2.$$

The jth output polynomial is a linear combination of the contributions from each of the k inputs and is given by

$$v^{(j)}(D) = \sum_{i=1}^{k} u^{i}(D) g^{(i,j)}(D). \tag{8.23}$$

Example 8.7

Consider again the $(4, 3, 2)$ code with input sequence $\boldsymbol{u}^{(1)} = (1\ 0\ 1)$, $\boldsymbol{u}^{(2)} = (1\ 1\ 0)$, and $\boldsymbol{u}^{(3)} = (0\ 1\ 1)$. The corresponding input polynomials are

$$u^{(1)}(D) = 1 + D^2$$
$$u^{(2)}(D) = 1 + D$$
$$u^{(3)}(D) = D + D^2.$$

Using eqn 8.23 and the generator polynomials $g^{(i,j)}(D)$ already derived, the first output polynomial is

$$\begin{aligned} v^{(1)}(D) &= u^{(1)}(D)g^{(1,1)}(D) + u^{(2)}(D)g^{(2,1)}(D) + u^{(3)}(D)g^{(3,1)}(D) \\ &= (1 + D^2)1 + (1 + D)0 + (D + D^2)0 \\ &= 1 + D^2 \end{aligned}$$

and the second

$$\begin{aligned} v^{(2)}(D) &= u^{(1)}(D)g^{(1,2)}(D) + u^{(2)}(D)g^{(2,2)}(D) + u^{(3)}(D)g^{(3,2)}(D) \\ &= (1 + D^2)1 + (1 + D)(1 + D) + (D + D^2)D \\ &= D^2 + D^3. \end{aligned}$$

Likewise $v^{(3)}(D)$ and $v^{(4)}(D)$ can be determined so giving

$$v^{(1)}(D) = 1 + D^2$$
$$v^{(2)}(D) = D^2 + D^3$$
$$v^{(3)}(D) = 1 + D^2 + D^3 + D^4$$
$$v^{(4)}(D) = D^3 + D^4.$$

□

The output polynomial $v(D)$ has contributions from $v^{(1)}(D), v^{(2)}(D), \ldots, v^{(n)}(D)$ suitably delayed to take into account the order in which bits leave the encoder and it can be shown that

$$v(D) = \sum_{j=1}^{n} D^{j-1} v^{(j)}(D^n). \tag{8.24}$$

The components of $\boldsymbol{v}$ are then given by the coefficients of $v(D)$.

Example 8.8
The output polynomial $v(D)$ for the $(4, 3, 2)$ code, given by eqn 8.24 is

$$v(D) = v^{(1)}(D^4) + Dv^{(2)}(D^4) + D^2v^{(3)}(D^4) + D^3v^{(4)}(D^4)$$

and using $v^{(1)}(D)$, $v^{(2)}(D)$, $v^{(3)}(D)$, and $v^{(4)}(D)$ obtained in Example 8.7 gives

$$\begin{aligned} v(D) &= (1 + D^8) + D(D^8 + D^{12}) + D^2(1 + D^8 + D^{12} + D^{16}) + D^3(D^{12} + D^{16}) \\ &= 1 + D^2 + D^8 + D^9 + D^{10} + D^{13} + D^{14} + D^{15} + D^{18} + D^{19}. \end{aligned}$$

Expressing this as the vector

$$v = (v_0\ v_1\ v_2\ v_3,\ v_4\ v_5\ v_6\ v_7,\ v_8\ v_9\ v_{10}\ v_{11},\ v_{12}\ v_{13}\ v_{14}\ v_{15},\ v_{16}\ v_{17}\ v_{18}\ v_{19})$$

gives

$$v = (1\ 0\ 1\ 0, 0\ 0\ 0\ 0, 1\ 1\ 1\ 0, 0\ 1\ 1\ 1, 0\ 0\ 1\ 1).$$

This therefore is the output given the inputs $u^{(1)} = (1\ 0\ 1)$, $u^{(2)} = (1\ 1\ 0)$, and $u^{(3)} = (0\ 1\ 1)$ and is the same as that obtained before, using convolution and using matrices. □

The output polynomial $v(D)$ can be expressed directly in terms of the input polynomials, substituting eqn 8.23 into eqn 8.24 gives

$$v(D) = \sum_{i=1}^{k} u^{(i)}(D^n)g^{(i)}(D) \tag{8.25}$$

where

$$g^{(i)}(D) = \sum_{j=1}^{n} D^{j-1}g^{(i,j)}(D^n) \tag{8.26}$$

is the generator polynomial of the ith input summed over all n outputs.

Example 8.9
The $(4, 3, 2)$ code has generator polynomials $g^{(1)}(D)$, $g^{(2)}(D)$, and $g^{(3)}(D)$ representing the individual contributions of the three inputs to the total output from the encoder. From eqn 8.26 we get

$$\begin{aligned} g^{(1)}(D) &= g^{(1,1)}(D^4) + Dg^{(1,2)}(D^4) + D^2g^{(1,3)}(D^4) + D^3g^{(1,4)}(D^4) \\ g^{(2)}(D) &= g^{(2,1)}(D^4) + Dg^{(2,2)}(D^4) + D^2g^{(2,3)}(D^4) + D^3g^{(2,4)}(D^4) \\ g^{(3)}(D) &= g^{(3,1)}(D^4) + Dg^{(3,2)}(D^4) + D^2g^{(3,3)}(D^4) + D^3g^{(3,4)}(D^4) \end{aligned}$$

and substituting the generator polynomials gives

$$\begin{aligned} g^{(1)}(D) &= 1 + D + D^2 + D^3 \\ g^{(2)}(D) &= D + D^3 + D^5 + D^6 \\ g^{(3)}(D) &= D^2 + D^3 + D^5 + D^{10} + D^{11}. \end{aligned}$$

If we once again consider the inputs $u^{(1)} = (1\ 0\ 1)$, $u^{(2)} = (1\ 1\ 0)$, and $u^{(3)} = (0\ 1\ 1)$, so that $u^{(1)}(D) = 1 + D^2$, $u^{(2)}(D) = 1 + D$ and $u^{(3)}(D) = D + D^2$, then using eqn 8.25 gives

$$\begin{aligned} v(D) &= u^{(1)}(D^4)g^{(1)}(D) + u^{(2)}(D^4)g^{(2)}(D) + u^3(D^4)g^{(3)}(D) \\ &= (1 + D^8)(1 + D + D^2 + D^3) + (1 + D^4)(D + D^3 + D^5 + D^6) \\ &\quad + (D^4 + D^8)(D^2 + D^3 + D^5 + D^{10} + D^{11}) \\ &= 1 + D^2 + D^8 + D^9 + D^{10} + D^{13} + D^{14} + D^{15} + D^{18} + D^{19} \end{aligned}$$

as obtained in Example 8.8. □

8.5 Graphical representation of convolutional codes

Convolutional codes can be represented graphically using tree, trellis and state diagrams, all of which show the state of a register and the encoder output for all possible inputs. The *state* of a register is defined as the contents of the stages at a given point during encoding. The diagrams provide an interesting way of looking at the structure and encoding of convolutional codes, and furthermore, the trellis diagram plays an important role when decoding using the Viterbi algorithm (considered in Section 8.6).

We first consider *state diagrams*. Figure 8.6 shows an encoder for a (2, 1, 2) convolutional code. At any point during encoding, the encoder can be in one of the 4 states 00, 01, 10 and 11, referred to as states S_0, S_1, S_2, and S_3 respectively. The states are shown in Fig. 8.7 as *nodes* (filled circles) connected together by *branches* (solid or dashed lines) that represent transitions from one state to another, depending upon whether the input is 1 (solid line) or 0 (dashed line). The output occurring with each transition is shown next to the relevant branch. The state diagram gives a complete description of encoding in that for a given input the output and the next state of the encoder can be determined. To arrive at the state diagram we need to consider the operation of the encoder, taking in to account all possible transitions.

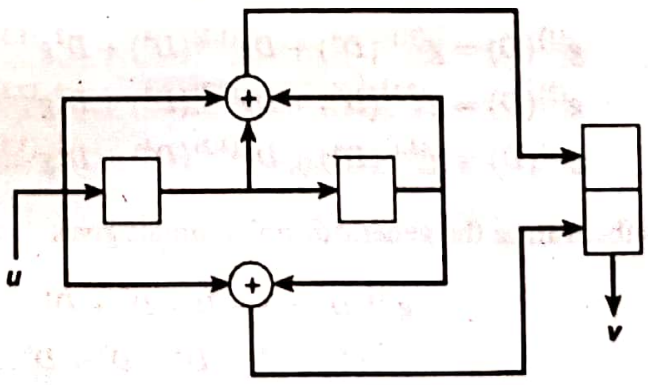


Fig. 8.6 A (2, 1, 2) convolutional encoder.

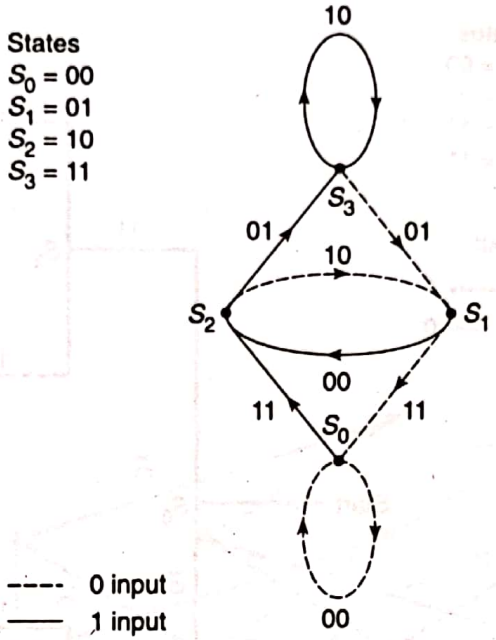


Fig. 8.7 State diagram for the (2, 1, 2) convolutional code.

Referring to Fig. 8.7 we assume that encoding starts at S_0, that is at the zero state with the stages at 00. Whilst the input stays at 0, transitions occur to the same state S_0 and the output is 00. At the first nonzero input, the stage contents change to 10 and the transition is to S_2 giving an output of 11. If the next input is also a 1, then the stage contents change to 11, the transition is to S_3 and the output is 01. Otherwise a 0 input gives a transition to S_1 and an output of 10. The state diagram is completed by considering transitions from S_3 and S_1 for inputs 0 and 1. Given an arbitrary input $\boldsymbol{u} = (u_0\, u_1\, u_2 \ldots)$ the output can be determined by following the transitions through the state diagram.

Example 8.10

Consider the input sequence $\boldsymbol{u} = (1\ 1\ 0\ 1\ 0\ 0)$. Referring to the state diagram in Fig. 8.7, and starting from the state S_0 we get

Input 1, transition to S_2 giving an output 11
Input 1, transition to S_3 giving an output 01
Input 0, transition to S_1 giving an output 01
Input 1, transition to S_2 giving an output 00
Input 0, transition to S_1 giving an output 10
Input 0, transition to S_0 giving an output 11

The output sequence is therefore $\boldsymbol{v} = (11, 01, 01, 00, 10, 11)$ □

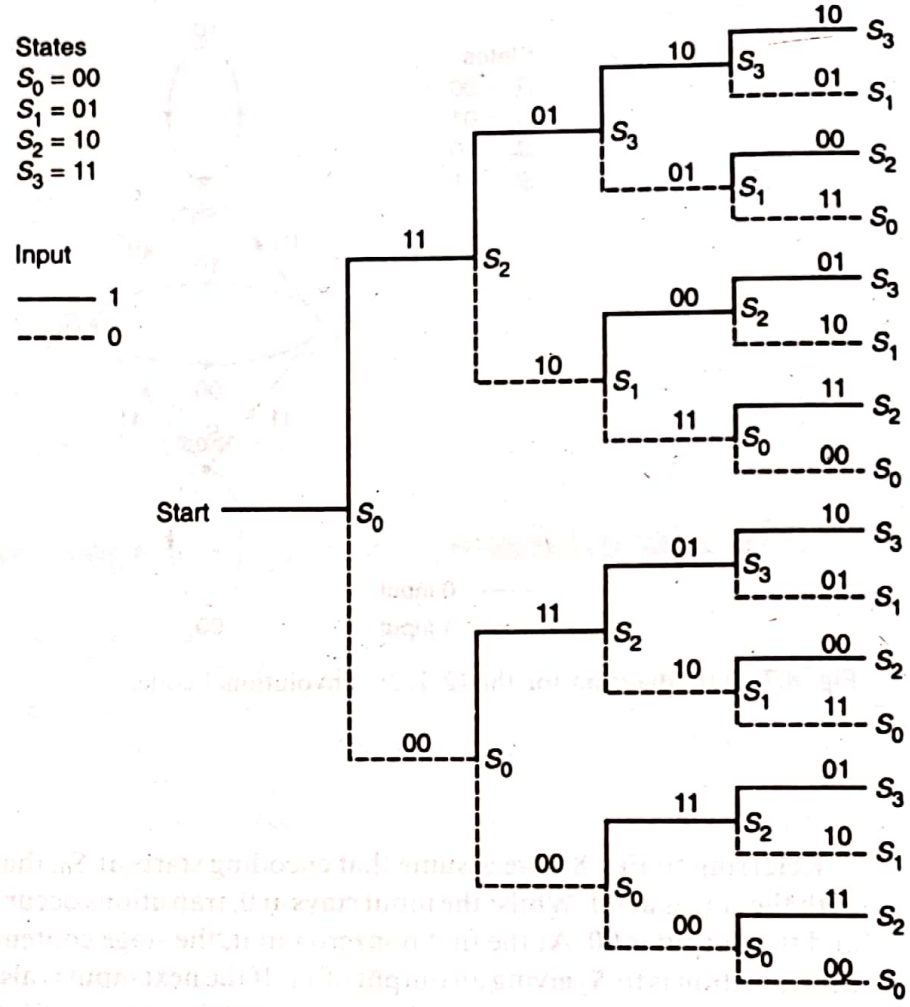


Fig. 8.8 Tree diagram for the (2, 1, 2) convolutional code.

Figure 8.8 shows a *tree diagram* for the (2, 1, 2) code. This again shows transitions from state to state, with corresponding outputs, for all inputs. The state and tree diagrams differ in that the latter shows the evolution of encoding with time. In the state diagram there is no way of knowing how a state was arrived at, from any specified state it is only possible to determine the next state and output for a given input. In the tree diagram moving from left to right represents the flow of time as the input sequence enters the encoder. At any node transitions can only occur to one of the two nodes to the right of the node. The output at each transition is shown above the horizontal portion of each branch. Encoding starts at the far left at S_0 and progresses from left to right. The tree diagram shown in Fig. 8.8 has been truncated after the 4th input, but theoretically the tree extends to the right infinitely.

A tree diagram gives an interesting graphical view of a convolutional code but is of limited practical use due to its size. However tree diagrams have a repetitive structure that allow the diagrams to be simplified whilst maintaining the basic feature of showing how encoding progresses with time. The resulting diagrams are known as *trellis diagrams* and Fig. 8.9 shows the trellis diagram for the (2, 1, 2) code. The trellis

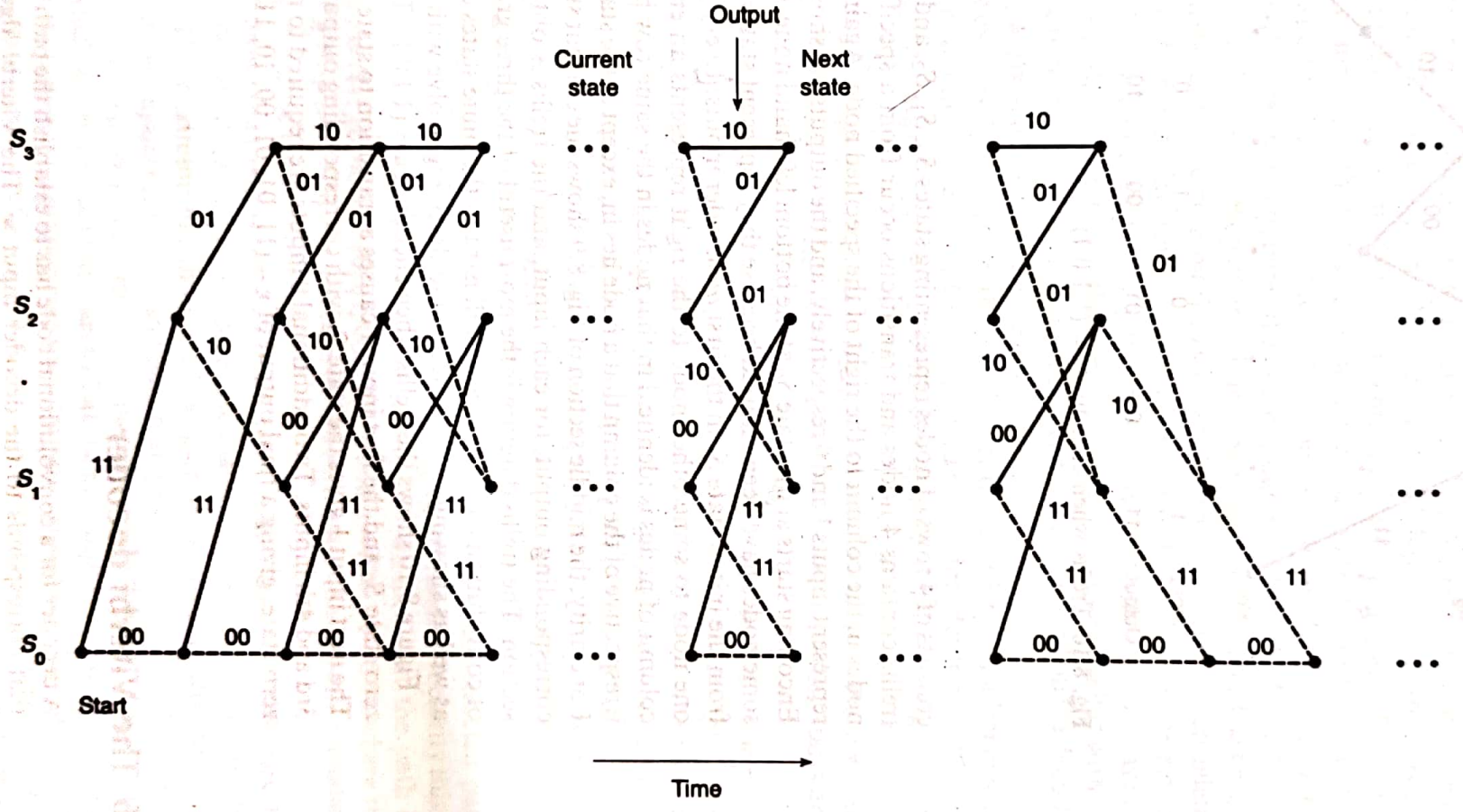


Fig. 8.9 Trellis diagram for the (2, 1, 2) convolutional code.

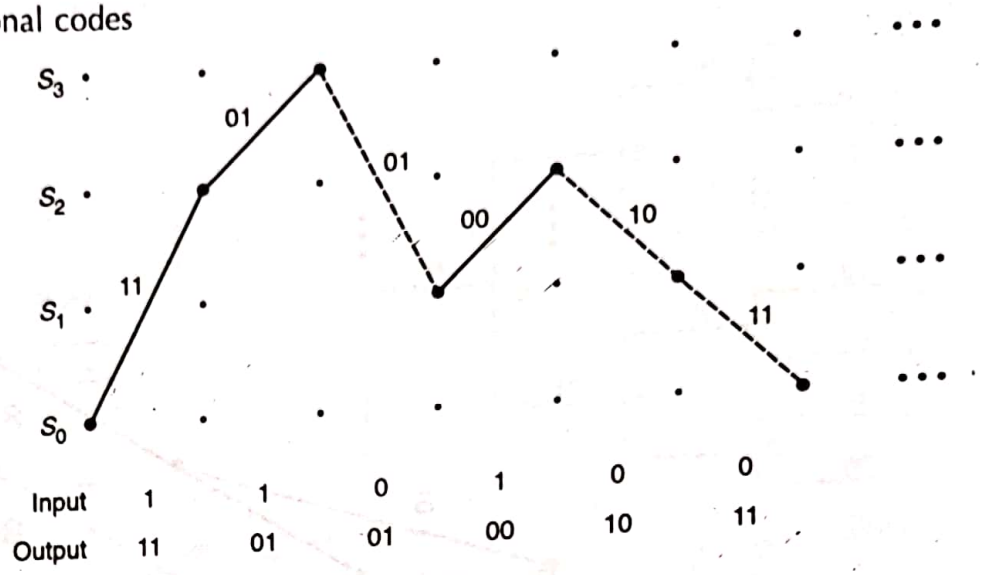


Fig. 8.10 Trellis when encoding $u = (1\ 1\ 0\ 1)$.

consists of 4 rows of nodes representing states S_0, S_1, S_2, and S_3. Each column in the trellis contains 4 nodes and transitions occur from a specified node to one of two nodes in the column to the right of the specified node. Again solid and dashed lines represent inputs 1 and 0 respectively, and the outputs are shown next to each branch. Encoding starts at the state S_0 at the bottom left-hand node. On the left of the trellis, some nodes have been omitted because they represent states that cannot be reached from the initial state S_0 in the first m encoder inputs. Every path in the trellis, from one node to some other node to the right represents an encoded sequence. Each column of nodes is identical in that nodes in the same row have the same branches irrespective of the column that a node lies in, except at the start and end of the trellis. For clarity, the middle section of Fig. 8.9 shows the current state and next state, with corresponding output, for each input, and the trellis is obtained by repeating this section. The trellis diagram is the most useful of the three graphical representations of convolutional codes. It clearly illustrates the finite states of a convolutional code as well as showing how encoding and decoding evolve with time.

Figure 8.10 shows the trellis when encoding $u = (1\ 1\ 0\ 1)$. The encoder starts at the zero state S_0 and the first input 1 causes a transition to state S_2 and an output of 11. The next input 1 gives the state S_3 with corresponding output 01 and so forth for the 3rd and 4th inputs. Two additional 0 inputs are required to return the encoder to its zero state, giving a final output of $v = (11, 01, 01, 00, 10, 11)$.

8.6 The Viterbi decoder

A decoder for a convolutional code has to establish the path through the code trellis that corresponds to the decoder input w. The Viterbi algorithm is a maximum-likelihood decoder that determines the path that w most likely corresponds to. This path is then taken to be the encoder output v, from which the encoder input u can be

metric which, for a binary-symmetric channel, is the Hamming distance between the branch output and the corresponding decoder input. Each path in the trellis is also assigned a metric that is the sum of the branch metrics forming the path. The aim of the decoder is to find the path that is the least distance away from the decoder input w. Figure 8.11 shows the metrics for the (2, 1, 2) code with input (00, 11, 01, 10, 10). The branch metrics are shown in parenthesis alongside each branch. At each node two paths arrive and the smallest path metric is shown next to the node. The branch of the path with the larger metric is marked with an X to show that it has been deleted and is no longer of interest. The other branch and the path, with the lower metric, are referred to as the *survivor* branch and path respectively. If the path metrics are the same then either path can be taken as the survivor. The node is then labelled with a T to show that the path metrics are tied. Note that metric values depend on the decoder input and are not fixed attributes of a trellis.

As decoding evolves the only paths that survive are those with the smallest metrics and these are stored by the decoder. On completion of the decoder input, the path starting at S_0 and ending at S_0 going backwards through the trellis, from right to left, is the required maximum-likelihood path. This path can be uniquely followed when going backwards through the trellis because there is only one branch, the survivor, feeding forward into each node. Recall that when encoding the encoder starts and ends at the state S_0 and the decoder must likewise do so.

Figure 8.12 shows the trellis resulting from a Viterbi decoder with input $w = (11, 10, 00, 01, 10, 01, 00, 10, 11)$ for the (2, 1, 2) code. Here w is the decoder input for an encoder output v that has incurred no errors, so $v = w$. Furthermore v is the encoded sequence for $u = (1\,0\,1\,1\,1\,0\,1)$. As in Fig. 8.11 the path and branch metrics are shown along with the deleted branches, for each decoder input. At the end of the input the decoder starts at the state S_0 and follows the path backwards along the surviving branches. The dotted path shows the resulting maximum-likelihood path. Note that at each node in the dotted path the path metric is zero, this is to be expected as w contains no errors. To determine the decoder output, that the constructed path corresponds to, we need to know the decoder output at each branch, for clarity this is not shown in Fig. 8.12. However, referring to Fig. 8.9 we find that the path in Fig. 8.12 gives $v = (11, 10, 00, 01, 10, 01, 00, 10, 11)$ and $u = (1\,0\,1\,1\,1\,0\,1)$, where the last two 0s of u (for returning the encoder to its zero state) have been excluded. Decoding has therefore been successful.

Figure 8.13 shows the trellis when $v = (11, 10, 00, 01, 10, 01, 00, 10, 11)$ incurs the 2-bit error $e = (00, 00, 01, 00, 00, 10, 00, 00, 00)$ so that $w = v + e = (11, 10, 01, 01, 10, 11, 00, 10, 11)$. The maximum-likelihood path is again shown by a dotted line. Note that for the first two inputs the branch and path metrics are 0, but on the third input (which contains the first error) they rise to 1. The path metric stays at 1 for the next two inputs (which are error free) but rises to 2 at the next input (which contains the second error). As there are no more errors the final path metric is 2, which means that all other paths through the trellis are at a distance of 2 or greater away from w. The dotted path shown in Fig. 8.13 is the correct path for v (see Fig. 8.12) and therefore decoding has been successful, despite the incurred errors.

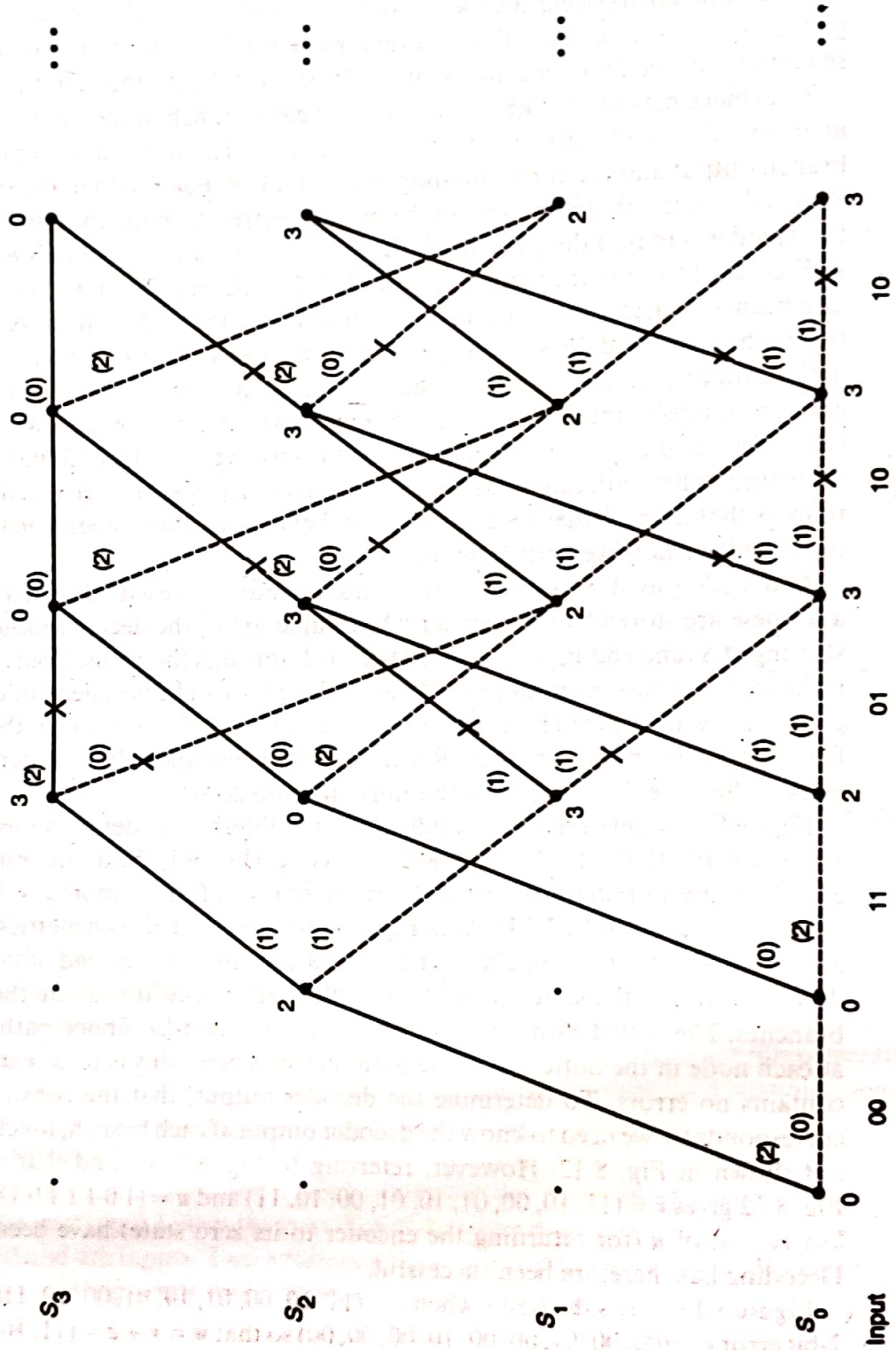


Fig. 8.11 Metrics and survivors in a trellis.

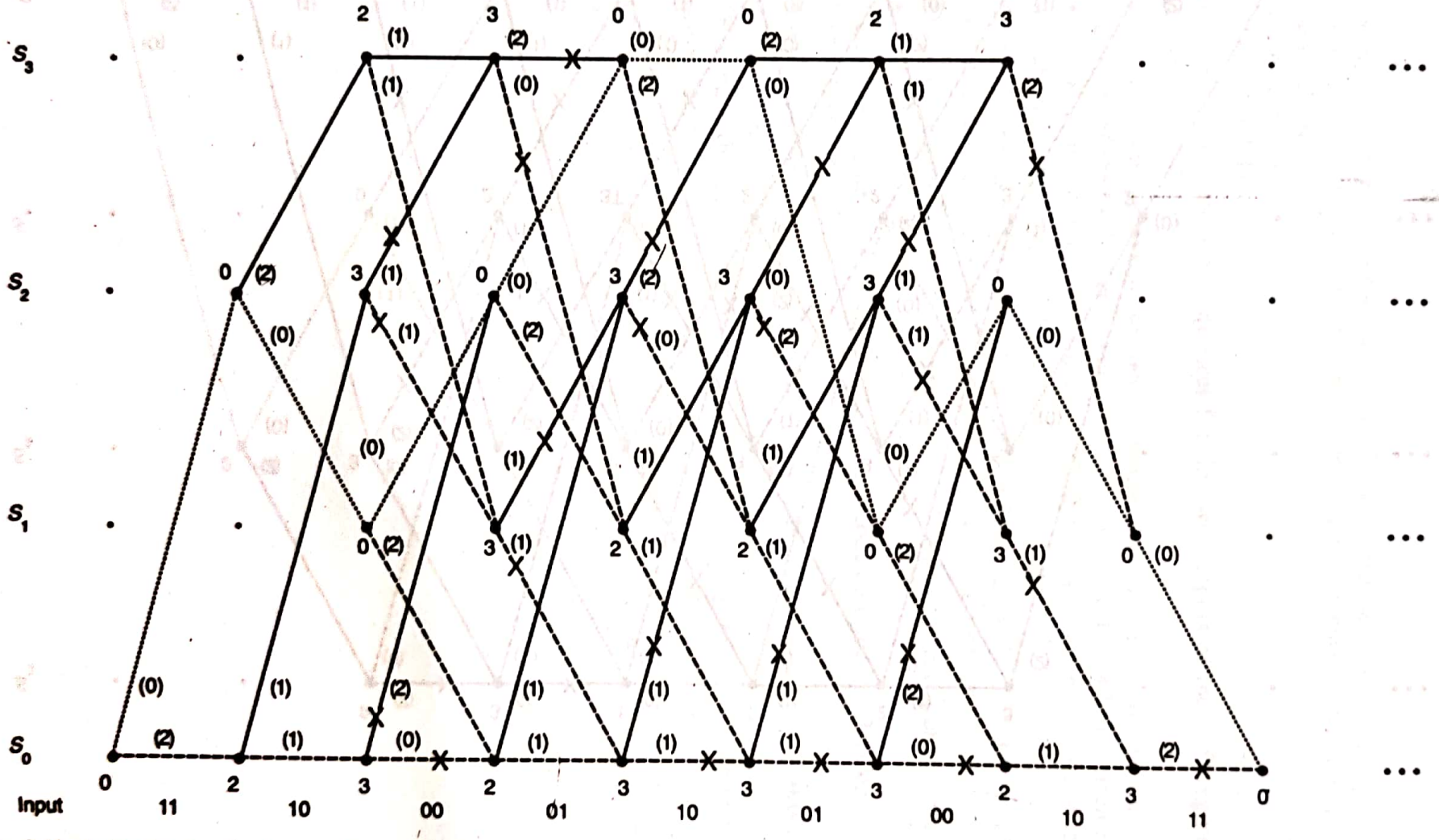


Fig. 8.12 Trellis for decoder input with no errors.

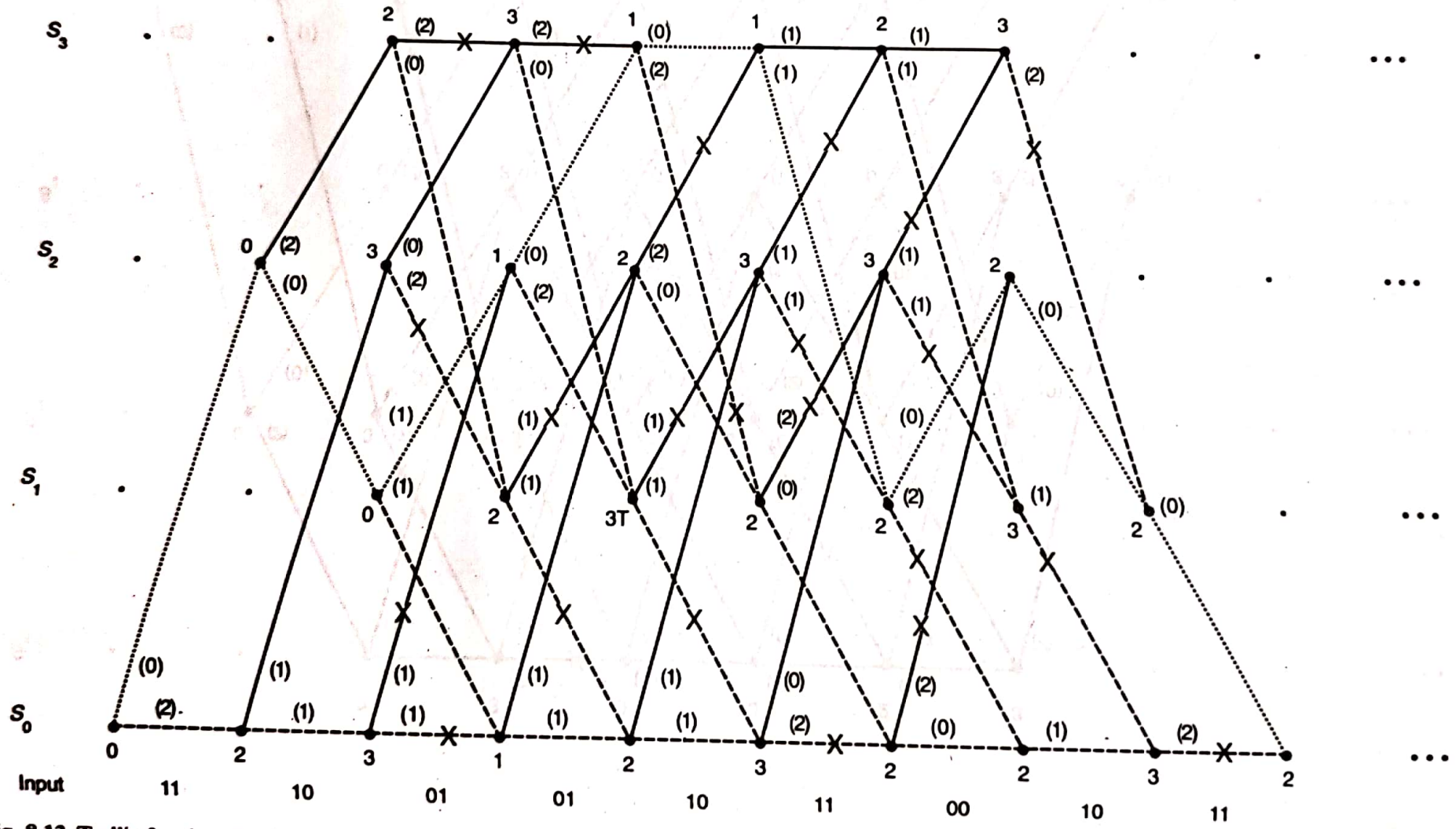


Fig. 8.13 Trellis for decoder input with 2 errors.

Problems

8.1 Determine the generator sequence of the shift register shown in Fig. 8.2. Hence, using convolution, determine the output sequences when the input sequences are
(i) (1 1 0 1 ...);
(ii) (1 1 0 1 1 1).

8.2 Determine the generator sequences for the two outputs of the shift register shown in Fig. 8.3.

8.3 Show that the generator sequences connecting the input to the output sequences of the (4, 3, 2) convolutional code, shown in Fig. 8.5, are given by eqns 8.8. Hence, using eqns 8.10 and 8.12, show that rth components of the output sequences are given by eqns 8.13.

8.4 Given the inputs $\boldsymbol{u}^{(1)} = (0\ 1\ 1\ 0)$, $\boldsymbol{u}^{(2)} = (1\ 0\ 0\ 1)$, and $\boldsymbol{u}^{(3)} = (0\ 1\ 0\ 1)$ to the (4, 3, 2) code use eqns 8.13 to determine the output sequence $\boldsymbol{v}$.

8.5 Draw an encoder for the (2, 1, 3) convolutional code with generator sequences $\boldsymbol{g}^{(1)} = (1\ 0\ 1\ 1)$ and $\boldsymbol{g}^{(2)} = (1\ 1\ 1\ 1)$.

8.6 Determine a generator matrix for the (2, 1, 3) code described in Problem 8.5. Hence find the output $\boldsymbol{v}$ when $\boldsymbol{u} = (0\ 1\ 0\ 1)$.

8.7 Using the generator matrix for the (4, 3, 2) code find $\boldsymbol{v}$ when $\boldsymbol{u}^{(1)} = (1\ 0\ 0\ 1)$, $\boldsymbol{u}^{(2)} = (1\ 1\ 0\ 0)$, and $\boldsymbol{u}^{(3)} = (0\ 1\ 0\ 1)$.

8.8. Determine the generator polynomials corresponding to the 12 generator sequences connecting the input sequences to the output sequences of the (4, 3, 2) code. Therefore determine the generator polynomials $g^{(1)}(D)$, $g^{(2)}(D)$ and $g^{(3)}(D)$, and find the output $\boldsymbol{v}$ when $\boldsymbol{u}^{(1)} = (1\ 0\ 0\ 1)$, $\boldsymbol{u}^{(2)} = (1\ 1\ 0\ 0)$, and $\boldsymbol{u}^{(3)} = (0\ 1\ 0\ 1)$.