Lecture 13

Goals

- Be able to encode using a convolutional code
- Be able to decode a convolutional code received over a binary symmetric channel or an additive white Gaussian channel

Convolutional Codes

Convolutional codes are an alternative way of introducing redundancy into the data stream. They are preferred to block codes in many situations because of the ease with which soft decision decoding can be performed. They are called convolutional code because the encoder output can be written as the convolutional of the encoder input with a generator sequence.

A convolutional code consists of k shift registers of length M or less each. The n outputs are linear combinations of the contents of the shift registers and the input. A convolutional code is describe by the memory length M (or constraint length $K = M + 1$), the number of input bits k the number of output bits n and the connections between the shift registers and the outputs.

Example (K=3,M=2, rate 1/2 code)

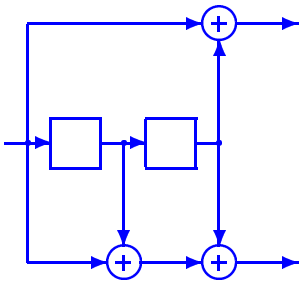


Figure 93: Encoder for rate 1/2 constraint length 3 convolutional code.

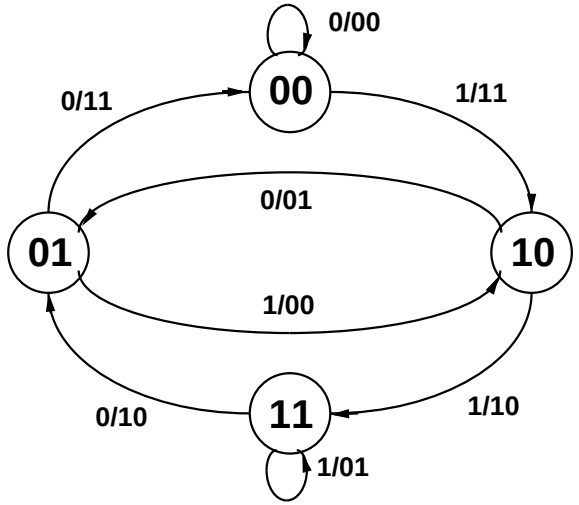


Figure 94: State Diagram for rate 1/2 constraint length 3 convolutional code.

We can describe the sequence of states the encoder passes through in time via a trellis diagram. This will be useful for decoding purposes.

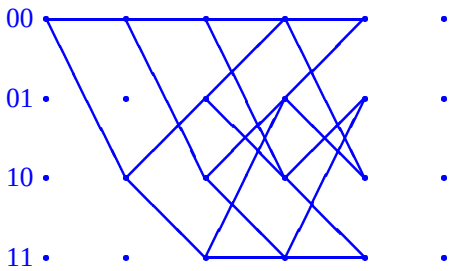


Figure 95: Trellis Diagram for rate 1/2 constraint length 3 convolutional code.

Decoding Convolutional Codes

Consider a state diagram for a particular convolutional code. Let x_m be the sequence representing the state at time m. Let x_0 be the initial state of the process ($p(x_0) = 1$). Later on we will denote the states by the integers 1,2,...,N. Since this is a Markov process we have that

$$p(x_{m+1}|x_m, x_{m-1}, ..., x_1, x_0) = p(x_{m+1}|x_m)$$

That is, the state at time $m+1$ depends only on the state at time m and not any previous state.

Let $w_m = (x_{m+1}, x_m)$ be the state transition at time m. There is a one-to-one correspondence between state sequences and transition sequences. By some mechanism (e.g. a noisy channel) a noisy version $\{z_m\}$ of the state transition sequence is observed. Based on this noisy version of $\{w_m\}$ we wish to estimate the state sequence $\{x_m\}$ or the transition sequence w_m. Since $\{w_m\}$

and $\{x_m\}$ contain the same information we have that

$$p(\mathbf{z}|\mathbf{x}) = p(\mathbf{z}|\mathbf{w})$$

where $\mathbf{z} = z_0, z_1, ..., z_{M-1}$, $\mathbf{x} = x_0, x_1, ..., x_M$, and $\mathbf{w} = w_0, ..., w_{M-1}$. If the channel is memoryless then we have that

$$p(\mathbf{z}|\mathbf{w}) = \prod_{m=0}^{M-1} p(z_m|w_m)$$

So given an observation $\mathbf{z}$ find the state sequence $\mathbf{x}$ for which the a posteriori probability $p(\mathbf{x}|\mathbf{z})$ is largest. This minimizes the probability that we chose the wrong sequence.

Thus the optimum (minimum sequence error probability) decoder chooses $\mathbf{x}$ which maximizes $p(\mathbf{x}|\mathbf{z})$: i.e

$$\begin{aligned} \hat{\mathbf{x}} &= \text{argmax}_{\mathbf{x}} p(\mathbf{x}|\mathbf{z}) \\ &= \text{argmax}_{\mathbf{x}} p(\mathbf{x}, \mathbf{z}) \\ &= \text{argmin}_{\mathbf{x}} \{-\log p(\mathbf{x}, \mathbf{z})\} \end{aligned}$$

$$= \text{argmin}_{\mathbf{x}} \{-\log p(\mathbf{z}|\mathbf{x}) p(\mathbf{x})\}$$

Using the memoryless property of the channel we obtain

$$p(\mathbf{z}|\mathbf{x}) = \prod_{m=0}^{M-1} p(z_k|w_m)$$

and using the Markov property of the state sequence

$$p(\mathbf{x}) = \left[\prod_{m=0}^{M-1} p(x_{m+1}|x_m)\right] p(x_0)$$

Define $\lambda(w_m)$ as follows:

$$\lambda(w_m) = -\ln p(x_{m+1}|x_m) - \ln p(z_m|w_m)$$

Then

$$\hat{\mathbf{x}} = \text{argmin}_{\mathbf{x}} \{\sum_{m=0}^{M-1} \lambda(w_m)\}$$

This problem formulation leads to a recursive solution. The recursive solution

is called the Viterbi Algorithm by communication engineers and is a form of Dynamic Programming as studied by control engineers. They are really the same though.

VITERBI ALGORITHM

Let $\Gamma(x_m)$ be the length (optimization criteria) of the shortest (optimum) path to state x_m at time m. Let $\hat{\mathbf{x}}(x_m)$ be the shortest path to state x_m at time m. Let $\hat{\Gamma}(x_{m+1},x_m)$ be the length of the path to state x_{m+1} at time m that goes through state x_m at time m.

Then the algorithm works as follows.

Storage

m, time index,

$\hat{\mathbf{x}}(x_m)$, $x_m \in \{1,2,...,M\}$

$\Gamma(x_m)$, $x_m \in \{1,2,...,M\}$

Initialization

$m = 0$,

o $\hat{\mathbf{x}}(x_0) = x_0$

$\hat{\mathbf{x}}(x_m)$ arbitrary, $m \neq x_0$

$\Gamma(x_0) = 0$

$\Gamma(x_m) = \infty$, $m \neq 0$

Recursion

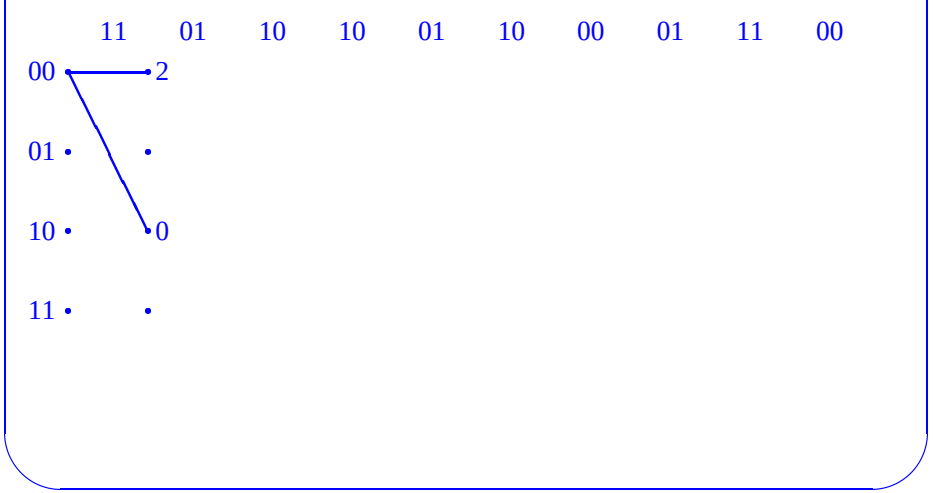
$\hat{\Gamma}(x_{m+1},x_m) = \Gamma(x_m) + \lambda(w_m)$

$\Gamma(x_{m+1}) = \min_{x_m} \hat{\Gamma}(x_{m+1},x_m)$ for each x_{m+1}

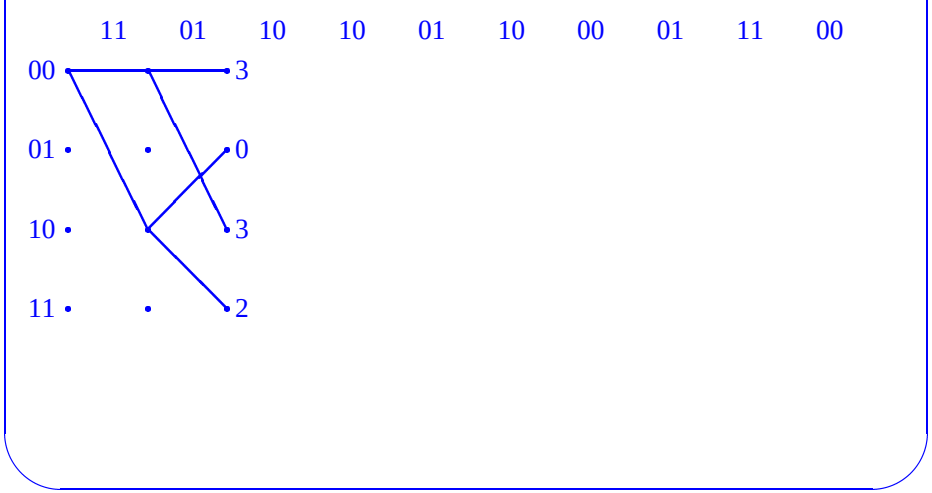
Let $\hat{\mathbf{x}}_m(x_{m+1}) = \text{argmin}_{x_m} \hat{\Gamma}(x_{m+1},x_m)$. $\hat{\mathbf{x}}(x_{m+1}) = \hat{\mathbf{x}}(x_m)\hat{\mathbf{x}}_m,(x_{m+1})$

Justification:

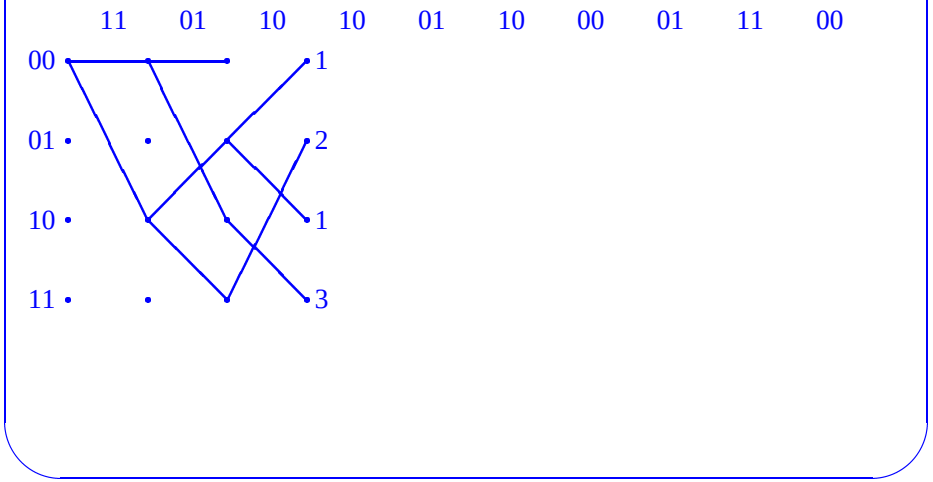
Basically we are interested in finding the shortest length path through the trellis. At time m we find the shortest length paths to each of the possible states at time m by computing all possible ways of getting to state $x_m = u$ from a state at time $m-1$. If the shortest path (denoted by $\hat{x}(u)$) to get to $x_m = u$ at time m goes through state $x_{m-1} = v$ at time $m-1$ (i.e. $\hat{\mathbf{x}}(u) = \hat{\mathbf{x}}(v),u$) then the corresponding path $\hat{\mathbf{x}}(v)$ to state $x_{m-1} = v$ must be the shortest path to state v at time $m-1$ since if there was a shorter path, say $\tilde{x}(v)$, to state v at time $m-1$ then the path $\tilde{\mathbf{x}}(v),u$ to state u at time m that used this shorter path to state v at time $m-1$ would be shorter then what we assumed was the shortest path). Stated another way if the shortest way of getting to state u at time m is by going through state v at time $m-1$ then the path used to get to state v at time $m-1$ must be the shortest of all paths to state v at time $m-1$.



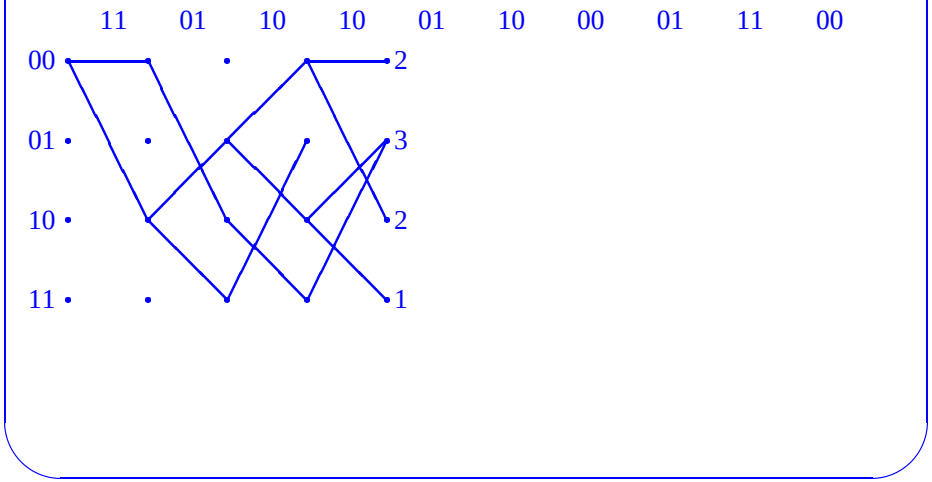
XIII-13



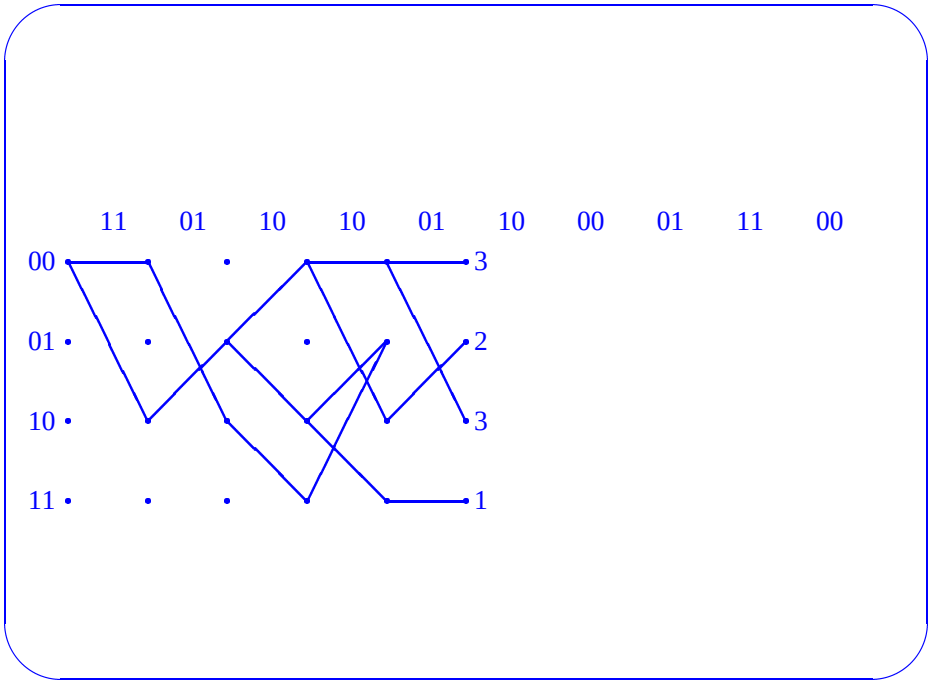
XIII-14



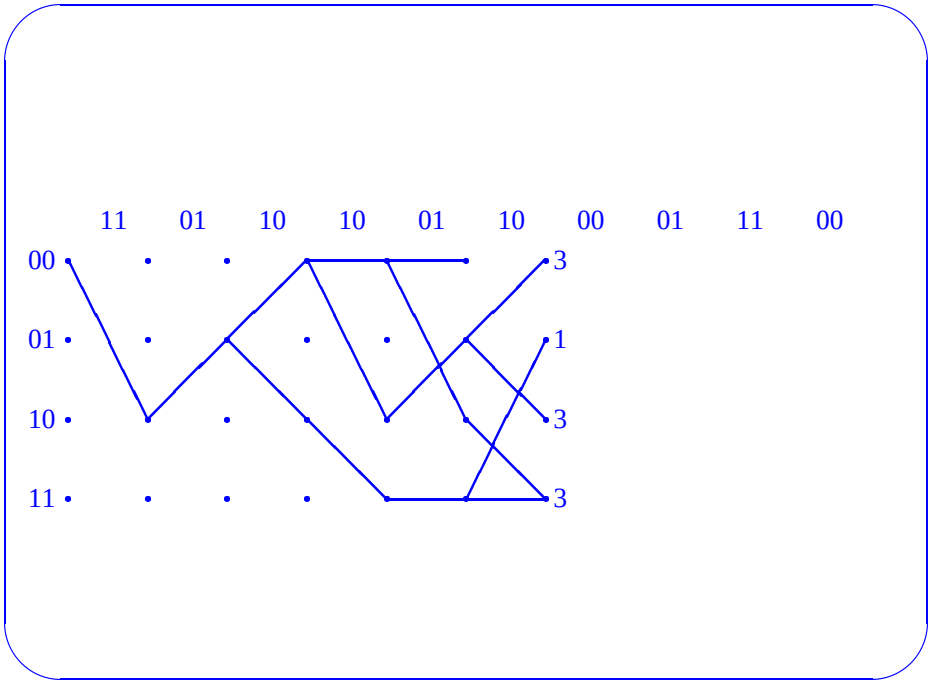
XIII-15



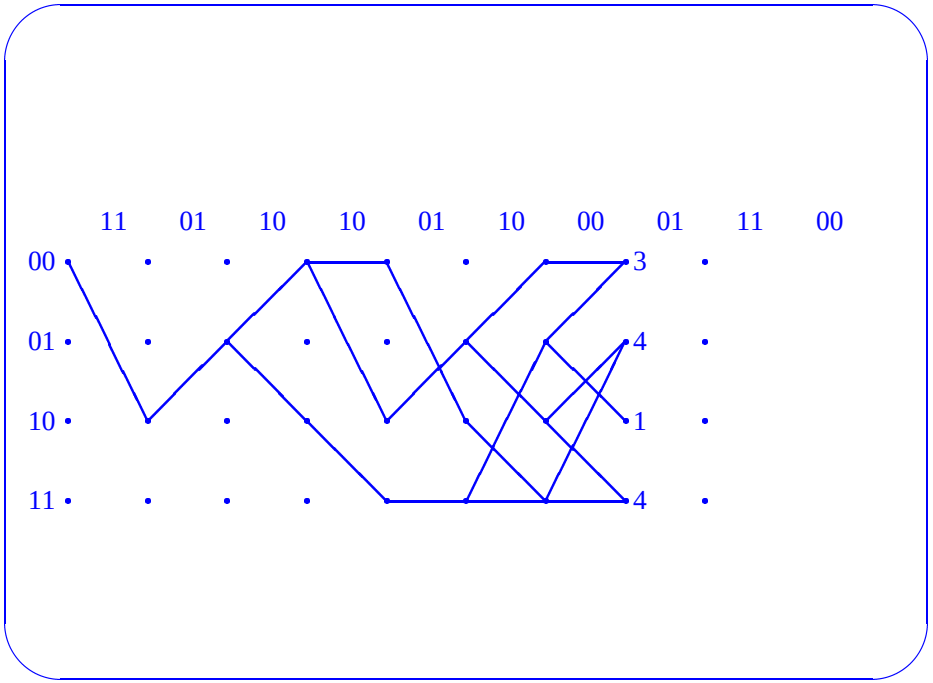
XIII-16



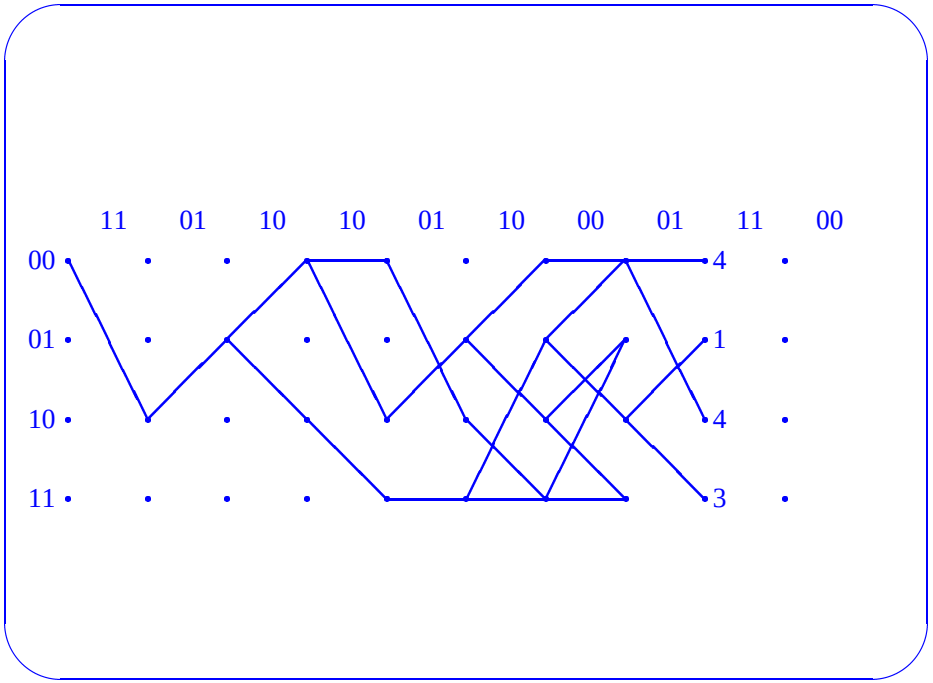
XIII-17



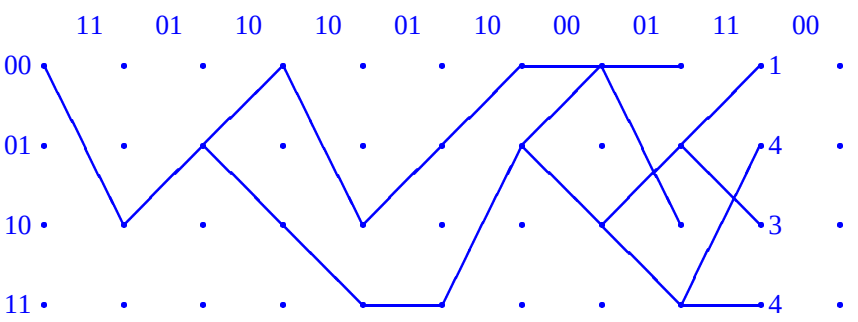
XIII-18

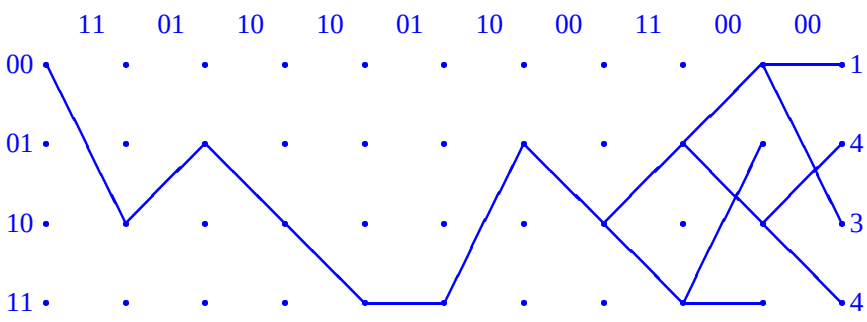


XIII-19



XIII-20





Error Bounds for Convolutional Codes

The performance of convolutional codes can be upper bounded by

$$P_b \leq \sum_{l=d_{free}}^{\infty} w_l D^l$$

where w_l is the average number of nonzero information bits on paths with Hamming distance l and D is a parameter that depends only on the channel. usually the summation in the upper bound is truncated to some finite number of terms.

Example 1

Binary Symmetric Channel crossover probability p.

$$D = 2\sqrt{p(1-p)}$$

Example 2

Additive White Gaussian Noise channel

$$D = e^{-E/N_0}.$$

Performance Examples

Generally hard decisions requires 2dB more signal energy than soft decisions for the same bit error probability. Also soft decisions is only about 0.25dB better than 8 level quantization.

Standard codes:

Example Convolutional Code 1:

Constraint length 7, memory 6, 64 state decoder, rate 1/2 has the following upper bound.

$$P_b \leq 36D^{10} + 211D^{12} + 1404D^{14} + 11633D^{16} + \cdots$$

There is a chip made by Qualcomm and Stanford Telecommunications that operates at data rates on the order of 10Mbits/second that will do encoding and decoding.

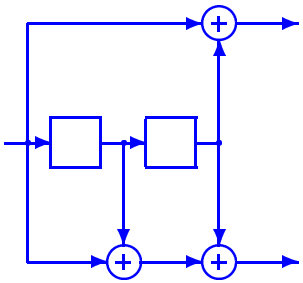
Example Convolutional Code 2: Constraint length 9, memory 8, 256 state decoder, rate 1/2

$$P_b \leq 33D^{12} + 281D^{14} + 2179D^{16} + 15035D^{18} + \cdots$$

Example Convolutional Code 3: Constraint length 9, memory 8, 256 state decoder, rate 1/3

$$P_b \leq 11D^{18} + 32D^{20} + 195D^{22} + 564D^{24} + 1473D^{26} + \cdots$$

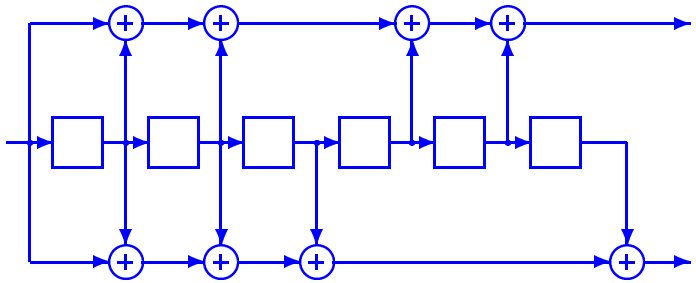
Example (K=3,M=2, rate 1/2 code)



This code has d_f of 5. The weight enumerator polynomial is

$$w(D) = \frac{D^5}{(1-2D)^2} = D^5 + 4D^6 + 12D^7 + 32D^8 + \cdots$$

Example ($K = 7, M = 6$, rate 1/2 code)



This code has d_f of 10. The weight enumerator polynomial for determining the bit error probability is given by

$$\begin{aligned} w(D) &= 36D^{10} + 211D^{12} + 1404D^{14} + 11633D^{16} + 77433D^{18} \\ &\quad + 502690D^{20} + 3322763D^{22} + 21292910D^{24} \end{aligned}$$

$$+ 134365911D^{26} + 843425871D^{28} + \cdots$$

We can upper bound the bit error probability by

$$P_b \le \sum_j w_j P_2(j) \le \sum_j w_j D^j = w(D)$$

The first bound is the union bound. It is impossible to exactly evaluate this bound because there are an infinite number of terms in the summation. Dropping all but the first N terms gives an approximation. It may no longer be an upper bound though. If the weight enumerator is known we can get arbitrarily close to the union bound and still get a bound as follows.

$$\begin{aligned} P_b &\le \sum_j w_j P_2(j) = \sum_{j=d_f}^{N} w_j P_2(j) + \sum_{j=N+1}^{\infty} w_j P_2(j) \\ &\le \sum_{j=d_f}^{N} w_j P_2(j) + \sum_{j=N+1}^{\infty} w_j D^j \end{aligned}$$

$$\begin{aligned} &= \sum_{j=d_f}^{N} w_j (P_2(j) - D^j) + \sum_{j=d_f}^{\infty} w_j D^j \\ &= \sum_{j=d_f}^{N} w_j (P_2(j) - D^j) + w(D) \end{aligned}$$

The second term is the Union-Bhattacharyya (U-B) bound. The first term is clearly less than zero, so we get something that is tighter than the U-B bound. By choosing N sufficiently large we can sometimes get significant improvements over the U-B bound.

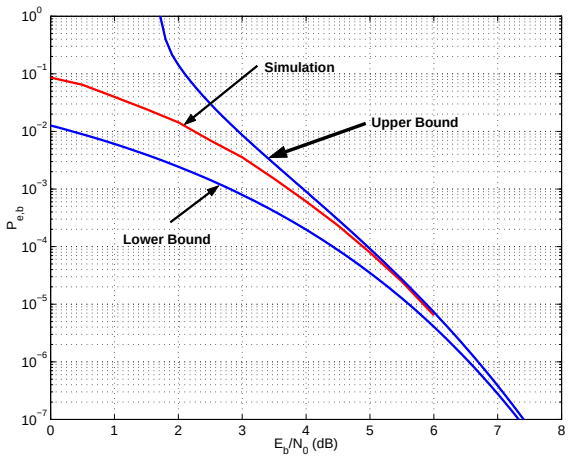


Figure 96: Error probability of constraint length 3 convolutional codes on an additive white Gaussian noise channel with soft decisions decoding (upperbound, simulation and lower bound).

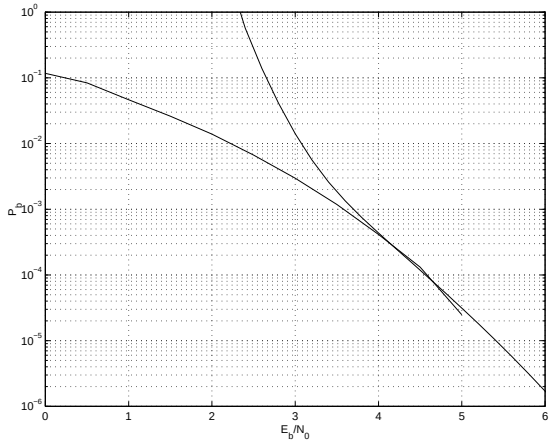


Figure 97: Error probability of constraint length 4 convolutional codes on an additive white Gaussian noise channel with soft decisions decoding (upperbound, simulation).

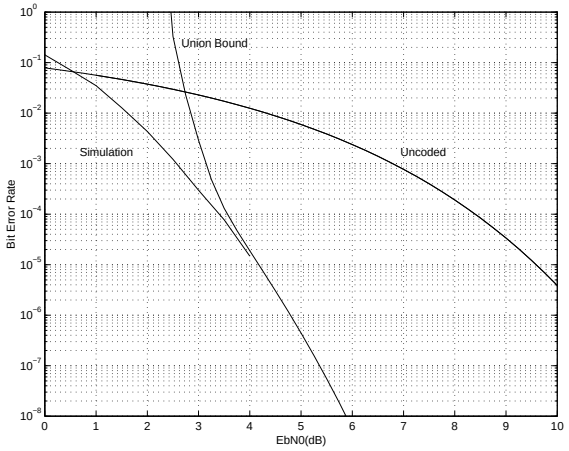


Figure 98: Error probability of constraint length 7 convolutional codes on an additive white Gaussian noise channel with soft decisions decoding (upperbound, simulation).

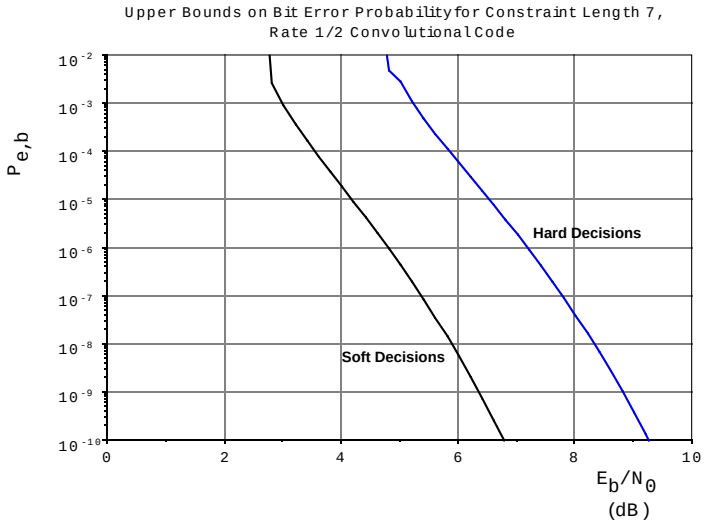


Figure 99: Error Probability of Constraint Length 7 Convolutional Codes on an Additive White Gaussian Noise Channel (hard and soft decisions).

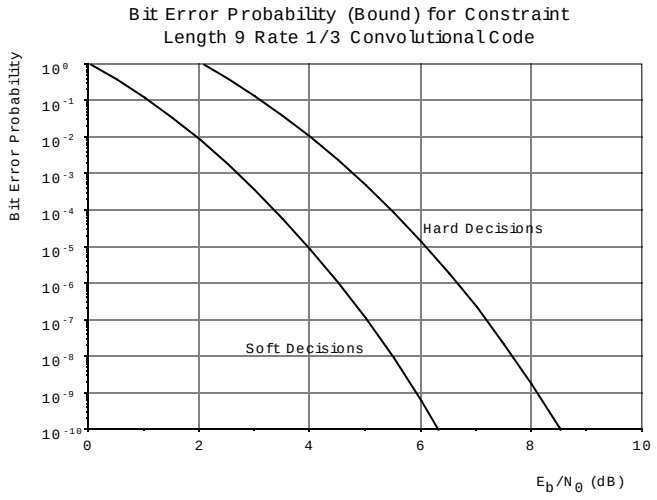
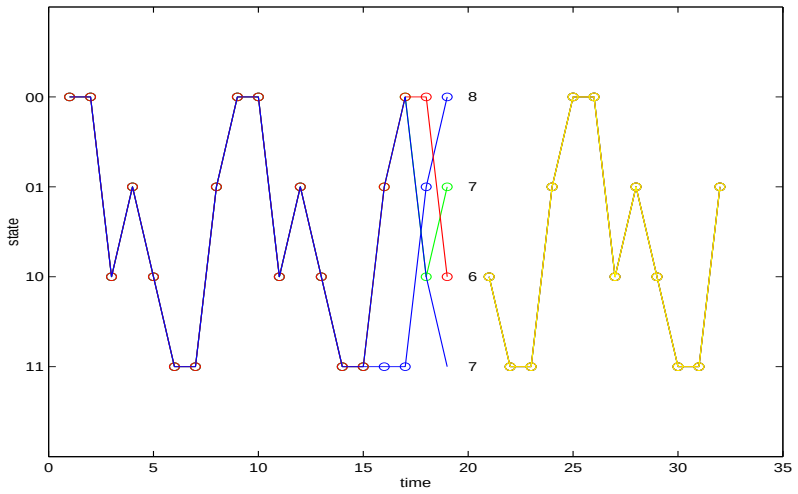
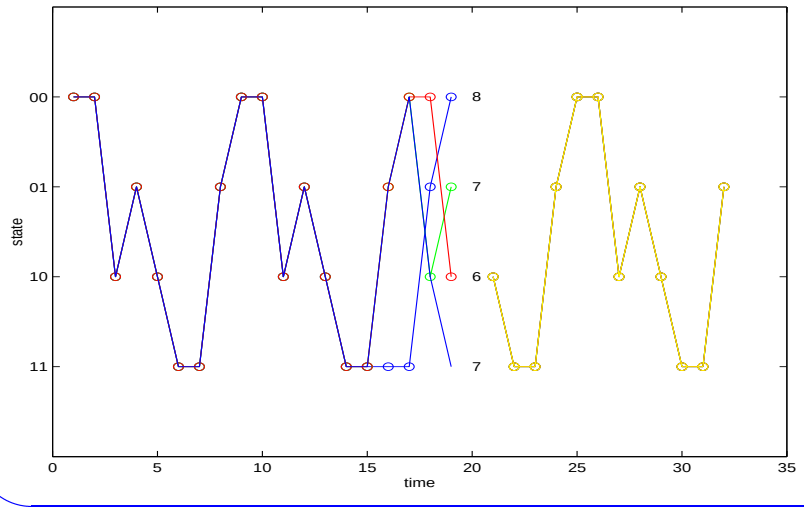


Figure 100: Error Probability of Constraint Length 9 Convolutional Codes on an Additive White Gaussian Noise Channel (hard and soft decisions).

Convolutional Decoder $p = 0.05$



Convolutional Decoder $p = 0.1$



Convolutional Decoder $p = 0.2$