EE/Ma 127b Error-Correcting Codes R. J. McEliece
draft of March 11, 2001 162 Moore

Details of Class Project #2
Due date: To be announced

You (and/or your team; maximum of four students per team) are expected to produce a computer program to implement the Viterbi decoding algorithm for the *Voyager* code, i.e., the $(2,1,6,10)$ binary convolutional code with generator matrix

$$(G_1(D), G_2(D)) = (1 + D^2 + D^3 + D^5 + D^6, 1 + D + D^2 + D^3 + D^6).$$

There will be *two* tests of your decoder, the "self-test," and the "demonstration" test. Both tests will require your decoder to perform on the BSC ("hard decisions") and the AWGN channel ("soft decisions").

• **The Self Test.** Here I want you to run experiments with your Viterbi decoder to produce a graph which shows the (approximate) relationship between E_b/N_0 and the decoded bit error probability for the given convolutional code, for E_b/N_0 ranging from 1 dB to 6dB, in increments of 0.5 dB.

• **The Demonstration.** At the time of your demonstration, I will ask you to encode N pseudorandom bits, then add Gaussian noise corresponding to a certain value of E_b/N_0, then decode the noisy bits using both "hard" and "soft" decisions, reporting in each case the number of decoded bit errors. I will not yet say how big N will be, but as discussed in class, I want you to truncate your survivors at length 32, outputting the oldest bit on the survivor with the best metric.

• **Important Fact:** For a binary code of rate R on the AWGN channel, the relationship between E_b/N_0, the *bit signal-to-noise ratio* and σ^2, the *Gaussian noise variance*, is given by

$$\sigma^2 = \left(2R\frac{E_b}{N_0}\right)^{-1},$$

so for example for a $R = 1/2$ code like the *Voyager* code, the relationship is simply

$$\sigma^2 = \left(\frac{E_b}{N_0}\right)^{-1}.$$

Finally remember that E_b/N_0 is always quoted in "dBs," where a dimensionless quantity x equals $10\log_{10} x$ dB's. Thus for example, a value of E_b/N_0 of 3.5 dB for the Voyager code corresponds to a value of $\sigma^2 = 0.4467$.

Additional details on Class Project 2.

1. Use the recursion
$$p_{n+6} = p_{n+1} \oplus p_n \qquad \text{for } n \geq 0$$
with the initial conditions
$$p_0 = 1, p_1 = p_2 = p_3 = p_4 = p_5 = 0,$$
to generate the N information bits. Ensure that the generated sequence is $100000100001\ldots$ and is periodic with period 63.

2. Encode the information sequence using the generator polynomials $G_1(D)$ and $G_2(D)$ given above.

3. The encoder outputs 0's and 1's. However, the input to the AWGN is ± 1. Therefore, map 0's to $+1$'s and 1's to -1's.

4. To simulate the AWGN, add the mean zero, variance σ^2 normal (Gaussian) random variables generated by the following segment of pseudo-code, to the $\pm 1's$ generated at the previous step. This program outputs two random variables, n_1 and n_2. Use n_1 (resp. n_2) for the encoder output corresponding to the generator polynomial $G_1(D)$ (resp. $G_2(D)$). `SEED` and σ (i.e., E_b/N_0) will be specified at the time of testing your program. `urand()` is a function which generates a random variable uniformly distributed in the interval $[0, 1]$.

```
main()
{
    ...
    global iurv;
    ...
    iurv = SEED;
    ...
    ...
}
normal(n1, n2, σ) /* See "Donald E.Knuth, The Art of Computer Programming, Vol.2,
p.104 " */
{
    do {
        x1 = urand();
        x2 = urand();
```

$x_1 = 2x_1 - 1;$

$x_2 = 2x_2 - 1;$

/* x_1 and x_2 are now uniformly distributed in [-1,+1] */

$s = x_1^2 + x_2^2;$

} `while` $(s \geq 1.0)$

$n_1 = \sigma x_1 \sqrt{-2 \ln s / s};$

$n_2 = \sigma x_2 \sqrt{-2 \ln s / s};$

}

`urand()`

{

`iurv` $= (14157$`iurv` $+ 6925)(\mathrm{mod} 32768);$

`return iurv`$/32767;$

}

5. To get the output of the BSC;
   (a) Take the sign of the output of the AWGN (Define Sign(0) = +1.)
   (b) Map +1's to 0's and -1's to 1's.
6. Truncate your survivors to length 32 and output the oldest bit on the survivor with the least metric ("Best State Decoding"). The number of the bits to be *decoded*, N, will be specified at the time of testing your program. To *decode* N *bits, generate* $N + 32$ *bits in* (1).

   Your program should output the fraction of decode bits in error (BER) in both cases.

   The following table lists some typical values.

N	σ	E_b/N_0	`SEED`	BER (AWGN)	BER (BSC)
1000	0.8	1.94dB	101	0.010	0.158
1000	0.9	0.92dB	111	0.107	0.225