COM 5140 Error-Correcting Codes — Chi-chao Chao
Fall 2019

Project No. 1
Due 5:00pm, December 3, 2019

You are expected to produce a program to implement the Viterbi decoding algorithm for the widely used $(2, 1, 6)$ convolutional code with generator matrix

$$\boldsymbol{G}(D) = (1 + D^2 + D^3 + D^5 + D^6 \quad 1 + D + D^2 + D^3 + D^6).$$

This code is assumed to be transmitted over an additive white Gaussian noise (AWGN) channel.

The deliverable will consist of three parts:

- **Part I, Demonstration.** At the time of demonstration, we will let you know: the number of decoded bits N, the bit signal-to-noise ratio (SNR) E_b/N_0 (in dB), the seed for the random number generator, and hard or (unquantized) soft decision. You should then report in each case the number of decoded bit errors made by your decoder and the corresponding bit error rate (BER). I want you to truncate your survivors at length 32, outputting the oldest bit on the survivor with the best metric.

- **Part II, Report.** You should run experiments with your Viterbi decoder to produce performance curves showing the relationships between E_b/N_0 (in dB) and the decoded BER (in logarithmic scale), with both hard-decision decoding, which corresponds to decoding on a binary symmetric channel (BSC), and unquantized soft-decision decoding, for E_b/N_0 ranging from 1 dB to 6 dB for hard decision and 1 dB to 4 dB for unquantized soft decision, with increments of 0.5 dB. Please also include your simulation data in tabular form, listing for each data point: the bit SNR E_0/N_0, the number of decoded bits, the number of decoded bit errors, and the BER. (These detailed data are only required for the two mandatory performance curves.) Please hand in before the deadline a *report* (in a hard copy) which includes, among other things, performance curves, and (optional) discussions of issues like output decision alternatives (best-state, fixed-state, majority-vote), survivor truncation length, etc. Your computer program *with comments* should be attached at the end of the report.

- **Part III, Program file.** You also need to submit, before the deadline, your program file. Please put all of your programs into a single file with your registration number as the file name, say, `105064851.c` or `105064851.cpp`. (If, after all kinds of attempts, you are still unable to put all of your programs in a single file, please compress your files into a single `rar` or `zip` file and use your registration number as the file name, say, `105064851.rar` or `105064851.zip`.) Upload your file to the iLMS system.

Additional Details on Project No. 1

1. Use the recursion

$$u_{l+6} = u_{l+1} \oplus u_l, \qquad \text{for } l \geq 0$$

with the initial conditions $u_0 = 1, u_1 = u_2 = u_3 = u_4 = u_5 = 0$ to generate the information bits. Ensure that the generated sequence is $100000100001\ldots$ and is periodic with period 63.

2. Encode the information sequence using the generator matrix $\boldsymbol{G}(D)$.

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

4. To simulate the AWGN channel with unquantized soft-decision decoding, add a normal (Gaussian) random variable of mean zero and variance σ^2 to the ± 1's generated at the previous step. For a binary code of rate R on the AWGN channel with antipodal signaling, the relationship between E_b/N_0 and σ^2 is given by

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

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

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

Please remember that E_b/N_0 is always quoted in "dBs," which equals $10\log_{10}(E_b/N_0)$. Thus for example, a value of E_b/N_0 of 4 dB for a $R = 1/2$ code corresponds to a value of $\sigma^2 = 0.3981$.

5. Use the following segment of pseudo code to generate normal random variables of mean zero and variance σ^2. The procedure `normal` outputs two independent normal random variables, n_1 and n_2, and `Ranq1` is a function which generates a random variable uniformly distributed in the interval $(0, 1)$.

```
unsigned long long SEED;
// SEED must be an unsigned integer smaller than 4101842887655102017.
unsigned long long RANV;
int RANI = 0;

main()
{
    ...
    ...
    ...
}

normal(n1, n2, σ)
```

```
{
    do{
        x_1 = Ranq1();
        x_2 = Ranq1();
        x_1 = 2x_1 - 1;
        x_2 = 2x_2 - 1;
        s = x_1^2 + x_2^2;
    } while (s ≥ 1.0)
    n_1 = σx_1 √(-2 ln s/s);
    n_2 = σx_2 √(-2 ln s/s);
}

double Ranq1()
{
    if ( RANI == 0 ){
       RANV = SEED ^ 4101842887655102017LL;
       RANV ^= RANV >> 21;
       RANV ^= RANV << 35;
       RANV ^= RANV >> 4;
       RANV = RANV * 2685821657736338717LL;
       RANI++;
    }
    RANV ^= RANV >> 21;
    RANV ^= RANV << 35;
    RANV ^= RANV >> 4;
    return RANV * 2685821657736338717LL * 5.42101086242752217E-20;
}
```

6. To get the output of the BSC, take the sign of the output of the AWGN channel and map $+1$'s to 0's and -1's to 1's.

7. In your decoder, truncate the survivors to length 32 and output the oldest bit on the survivor with the best metric. To decode N bits, generate $N + 31$ bits in (1). Finally compare the decoded information sequence with the original information sequence. If there are K bit errors, K/N will be a good estimate of the decoded BER.

8. As a partial check, some typical values are listed below.

E_b/N_0	BER (BSC)	E_b/N_0	BER (AWGN)
4.5 dB	2.1×10^{-3}	2.5 dB	2.2×10^{-3}
5.0 dB	6.4×10^{-4}	3.0 dB	5.3×10^{-4}

Other Notes for Demonstration

1. The survivor truncation length corresponds to the actual storage requirement of the survivors. For example, a survivor truncation length of 32 for this code means that each survivor stores 32 bits.

2. For the illustration below, suppose a state is described as the content of the feed-forward shift register in the encoder $\boldsymbol{s} = (s_1, s_2, s_3, s_4, s_5, s_6)$, where the input information bit first fed to s_1 and then shifted from left to right. In the trellis diagram, consider placing the states vertically from top to bottom in the order of (0 0 0 0 0 0), (1 0 0 0 0 0), (0 1 0 0 0 0), (1 1 0 0 0 0), (0 0 1 0 0 0), ..., (1 1 1 1 1 1). *What to do in case of tied metrics?* In the "add-compare-select" step the two metrics could be equal. In this case, if 0's and 1's are equally probable to occur in the transmitted information sequence, in principle you can safely select either case, and it will not affect the decoder performance. Yet for the purpose of demonstration, always choose the upper branch as the survivor. If best-state output decision is employed, in case of tied metrics, in principle you can also safely select either case, but again for the purpose of demonstration, always choose the survivor of the uppermost state.

3. Except in the procedure `normal` for generating noise, if a random number is needed in your program, use other random number generators instead of the function `Ranq1`, for the purpose of demonstration.

4. Each call of the procedure `normal` can return two independent normal random variables, n_1 and n_2. Please use both of them in your program. Specifically, since this is a $(2, 1)$ code, each branch transition consists of two encoded bits, say x_1 and x_2. Add n_1 and n_2 to x_1 and x_2, respectively, to get the two channel outputs y_1 and y_2, i.e., $y_1 = x_1 + n_1$ and $y_2 = x_2 + n_2$.