

Problem 6.1 *8.1 in Proakis*

Problem 6.2 *Reed-Solomon Code Exercise*

Consider the finite field with 5 elements: $0, 1, 2, 3, 4$ where the usual rules of addition and multiplication modulo 5 apply. (e.g. the multiplicative inverse of 2 is 3 and 4 is its own inverse, while the additive inverse of 1 is 4 and 2's is 3.)

Consider the Reed-Solomon code that encodes two symbols as a linear polynomial: $f_{f_0, f_1}(z) = f_0 + f_1z$ and uses the evaluations of this polynomial at all 5 elements of the field as the codewords.

- a. Write out the entire codebook explicitly. There are 25 codewords of length 5 each.
- b. What is the minimum distance between the codewords when viewed using the Hamming metric on an alphabet of size 5?
- c. Suppose that we are encoding the pair $(3, 1)$ and an erasure occurs on the second and fourth symbol in the codeword. Work through the erasure correction algorithm to show how we can decode correctly.
- d. Suppose that we are encoding $(4, 1)$ and the first symbol in the codeword is incorrectly received as a 3 rather than a 4. Work through the error correction algorithm given in class to show how we can decode correctly.
- e. Suppose that we are encoding $(5, 1)$ and the first symbol in the codeword is incorrectly received as a 3 rather than a 5 while the second symbol is erased. Show how we can still decode correctly to recover from both the error and the erasure.
- f. Use a computer if necessary to compute the minimum distance between the codewords if we view them as represented on a binary alphabet with three binary digits representing each symbol from 0-5. (So the codewords have length 15 bits and there are 25 codewords) Comment on the error correcting power of this code when viewed on these binary symbols. Is the distance distribution identical starting from each codeword?

Problem 6.3 *Getting to very low error probability in practical systems.*

One of the tricks that was mentioned in class was to nest codes by using the codewords of one as the basic symbols of another code. When the underlying probability of error is already low, this can also be done at the bit or byte level to avoid having to use giant finite fields.

Consider the following layered structure:

1. From the lower level, we have access to the semi-reliable transport of bits so that the probability of error on any individual bit is 10^{-4} and the error events on different bits are independent. (you can think of this as a virtual binary symmetric channel with $\epsilon = 0.0001$)

2. We group the incoming bits into 8 bit bytes and view these bytes as symbols from the finite field $GF(256)$.
3. We take 244 incoming symbols and construct a polynomial which is then evaluated at all 256 points in the finite field.
4. The resulting 256 coded bytes are sent across the virtual BSC on a bit-by-bit basis.
5. The received bits are grouped into 256 received bytes and then we decode the rate $(244/256 \approx 0.95)$ Reed-Solomon code to recover the original bytes.

What is the bit-error probability at the end of this process? (An upper bound on the probability is good enough)

Suppose that you wanted to get the probability of bit-error even lower, but did not want to do finite field arithmetic on anything bigger than 8-bit bytes. One obvious choice would be to decrease the rate of the Reed-Solomon code above to increase the number of byte errors that are corrected. Evaluate what happens if you do that.

What else could you do that can be thought of as a further step to the layers already described rather than changing the rate on the original Reed-Solomon? (Hint: think about using another byte-oriented Reed-Solomon code on top of it and what you would need to do in order to make it work) Compare with the simple decrease of rate strategy.

Problem 6.4 Convolutional Code (Computer Exercise)

Suppose that you are given a rate $1/N$ convolutional code (on a binary alphabet) with constraint length B in terms of N different “impulse responses” each of length B .

- a. How many different states could there be in a convolutional code with the parameters N and B ?
- b. Implement a convolutional encoder in whatever programming language or environment that you prefer that accepts as input the NB bits specifying the convolutional code and an L bit data string. It should output the $M = LN$ encoded bits to be transmitted assuming that we start at the 0 state. (Test it out to make sure it works)
- c. Implement an ML decoder (e.g. Viterbi) in whatever programming language or environment that you prefer that accepts as input the NB bits specifying the convolutional code and an $M = LN$ bit received string. It should output its best estimate for the $L = M/N$ bits originally transmitted. (Test it out to make sure it works)
- d. Generate a rate $1/3$ convolutional code with $B = 9$ at random using independent fair coin tosses. Generate an input of length 20 at random and then run it through the encoder. Simulate a BSC with $\epsilon = 0.1$ on the transmission to get your received sequence. Then run the decoder to get the estimate of the received sequence. Estimate through simulation the average probability of error on the different bit positions: 1, 2, 3, \dots , 20.