ECEN 455 - Digital Communications

Lab 2 - Source Coding

Fall 2015

Introduction

Source coding is important in communication systems as we are often limited by the capacity of the communication channel we are using. Given a fixed maximum data rate, we can transfer our file quicker if we could first compress it so that we need to transfer fewer bits over the channel. In this lab we are going to investigate a few of the various source coding (compression) techniques discussed in class. We will use a text file "constitution.txt" which fundamentally consists of a sequence of ASCII characters and then apply the Huffman coding technique to compress the file.

Task 1 - Measure the source entropy

First measure the entropy per source letter. To do so, measure the relative frequency of each letter that appears in the text file. Based on that probability distribution, estimate the source entropy in bits per source letter. You will need to read the text into MATLAB and then count the number of times each letter/symbol appears in the text file. From this symbol count you can compute the relative frequency of each symbol. Based on this probability distribution, compute the entropy (in bits per text letter) according to

$$H = -\sum_{k=0}^{127} p_k \log_2(p_k), \tag{1}$$

where the index k is the ASCII number corresponding to each letter. Note that several ASCII characters do not occur in the source text and as such will have a relative frequency of zero. MATLAB will not like it if you ask it

to compute $\log_2(0)$ so you will have to decide how to deal with this problem in your program.

Task 2 - Compress the source file using Huffman coding

In this section you will write a set of programs to compress (and uncompress) the source file using the Huffman coding technique. You will do so by completing the following subtasks:

A. Write a program `C=HuffmanCode(p)` whose input is a discrete probability distribution `p` and whose output is a cell array `C` consisting of the codewords assigned to each source letter. Test your program on a relatively simple probability distribution `p` that you can check by hand to be sure your program is working correctly. Report the results of your test and once you are sure your program functions correctly, run your program on the source distribution found in Task 1. Based on the code designed, determine the average number of bits per source symbol the Huffman code should need for the file "constitution.txt."

B. Write a program `EncodedBits=HuffmanEncode(SourceText,C)` that will use the Huffman code `C` that you previously designed to encode a sequence of source letters `SourceText` and encode them into a sequence of bits `EncodedBits`. As before test your program on a short string of text to be sure it is functioning correctly. Report the results of your test run and then once you are sure your program functions correctly, run your program on the source text in the file "constitution.txt." Note the length of your compressed file (in bits) and compare that with what you expected based on the average codeword length of your Huffman code calculated previously.

C. Write a program `DecodedText=HuffmanDecode(EncodedBits,C)` that will use the Huffman code `C` that you previously designed to decode a sequence of encoded bits `EncodedBits` into a sequence of source letters `DecodedText`. As before test your program on a short string of bits to be sure it is functioning correctly. Report the results of your test run and then once you are sure your program functions correctly, run your program on the encoded bits

that you obtained previously by compressing the file "constitution.txt." Compare the decoded text with the original text file and note whether or not there are any differences.

You may find some of the following MATLAB commands useful (use the MATLAB help command for documentation on what these do): `char`, `cell`, `double`, `fopen`, `fread`, `hist`, `dec2bin`.