Matthew Wilson (u0499184)

Analysis for Assignment 12: Huffman Compression

November 23, 2016

1. Timing Test Overview

Using a range of 50 ASCII values from '0' to 'a' (48 to 97), I filled a file with a constant number of characters and used that to test compression. I held the size of the file constant and varied the number of unique characters. I started with 5 unique characters ('0','1','2','3','4'), and added 5 unique characters for each test, until I got to 'a'. As the number of unique characters increased, the frequency of each character decreased to keep constant size.

Details

1. For the first test, choose 5 ASCII values. Generate a character array that has `frequency` number of each of the 5 ASCII values, where `frequency` is determined by dividing the overall size by the number of unique characters.
2. Loop through your ASCII values, adding 5 unique values for each test. I used the range 48 to 97 because it was a nice group of 50 values.
3. Once you create the character arrays, write these to files and compress the files using Huffman coding.
4. Record the size of the compressed file vs the original file.

Explanation of graph

The left graph shows the compression ratio ($\frac{compressed}{orignal}$) for increasing number of unique characters. It can be seen that the ratio gets higher (i.e. worse compression) as the number of unique characters increases. This makes sense because the Huffman Tree has to get bigger to support more unique characters. As the tree gets bigger, the codes get longer.

The right graph just shows the inverse relationship for frequency. Number of unique characters and frequency of those characters have an inverse relationship. As frequency of character increases, the ratio decreases (i.e. better compression).

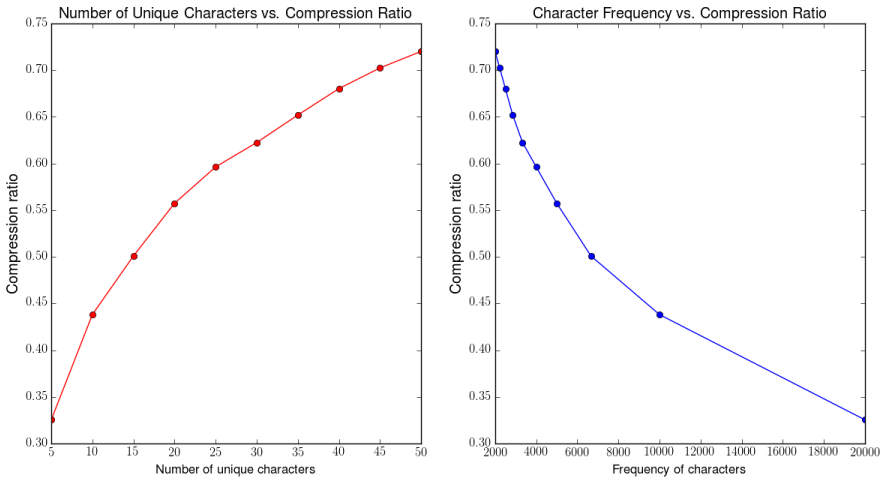

Figure 1: Huffman Compression Effectiveness

2. Huffman's algorithm will significantly compress files of text that are very uniform. One example of this is a list of phone numbers. These only have a few ASCII values, so the potential for compression is great.

Huffman's algorithm will work poorly on well distributed or random text. For writing, like this document, it wouldn't work as well because many different ASCII values are used. The worst compression would be for a random scramble of ASCII values. Although I suspect it is fairly rare to have a file that uses all of the ASCII characters and then you also have Unicode.

3. Huffman's algorithm merges the two smallest-weight trees because this causes them to be at the bottom of the tree. This is advantageous because the longer codes will get assigned to the least common characters. Shorter codes will be saved for the most common characters. This is the whole reason that Huffman compression works. The more frequent characters get shorter codes, and thus take up fewer bytes. This is a similar idea to Morse Code, where one of the most common letters: "i" is represented by a single dit, while another common letter: "t" is represented by a single dah. Less common characters have longer encodings.

4. Huffman's compression performs lossless compression. We know this because we don't lose part of our message when we compress and decompress. Lossy cmpresion woud not b great for important messa. It makes much more sense for images or videos where we can afford to lose resolution.

5. I spent about 8-9 hours on this assignment, probably.