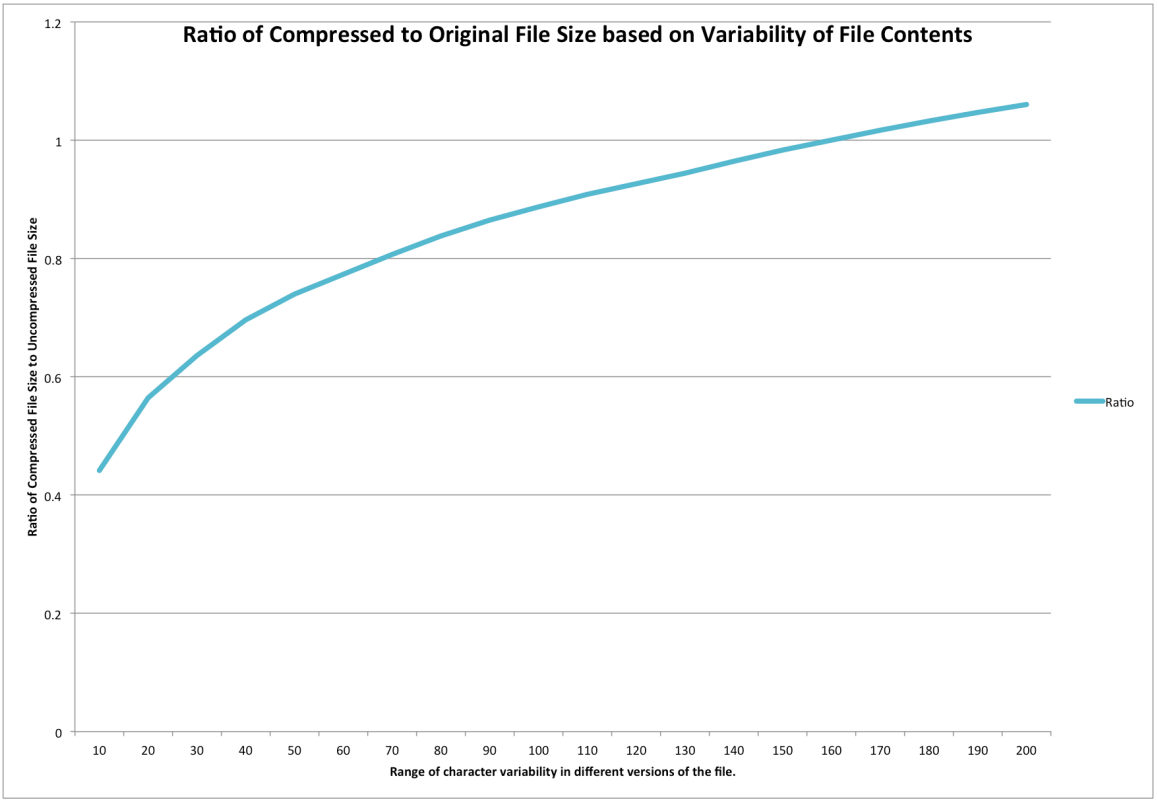
Alex Motro
CS 2420-005
HW 12

1. Design and conduct an experiment to evaluate the effectiveness of Huffman's algorithm. How is the compression ratio (compressed size / uncompressed size) affected by the number of unique characters in the original file and the frequency of the characters?

I created a file of a large size, I used 10,000. To this file, I added 10,000 random numbers by their ASCII value. At first they were in the range of 0 to 10, so that there were 10,000 digits that were in a small range, making for a lot of repetitions. I compressed the file and compared the lengths of the compressed file to the original file and took the ration. The same procedure was performed with a bigger range of numbers, I incremented it by 10, and the same ratio information was collected. I did this up to a range of 150 possible characters

Here are the results:



The results came out as expected. With smaller ranges came greater differences between compressed and uncompressed files. The more varied the file was, the less effective the compression was at saving space.

4. For what input files will using Huffman's algorithm result in a significantly reduced number of bits in the compressed file? For what input files can you expect little or no savings?

Files with a lot of repetitions benefit immensely from Huffman's algorithm. The more there is of a certain character, the fewer nodes there are in the trie. But with files

with lots of unique characters and a lot of variation, there still need to be nodes created for all of them, and there is not as much space saved.

5. Why does Huffman's algorithm repeatedly merge the two smallest-weight trees, rather than the two largest-weight trees?

This method leaves the largest weighted trees at the front of the priority queue, and since we're interested in the largest trees this is very useful.

6. Does Huffman's algorithm perform lossless or lossy data compression? Explain your answer. (A quick google search can define the difference between lossless and lossy compression).

This algorithm is lossless. The same file can be reconstructed if need be because Huffman's algorithm doesn't sacrifice data for size, it merely stores it in a different way to make it easier and quicker to use, but it keeps track of every little bit.

7. How many hours did you spend on this assignment?

3-5 hours