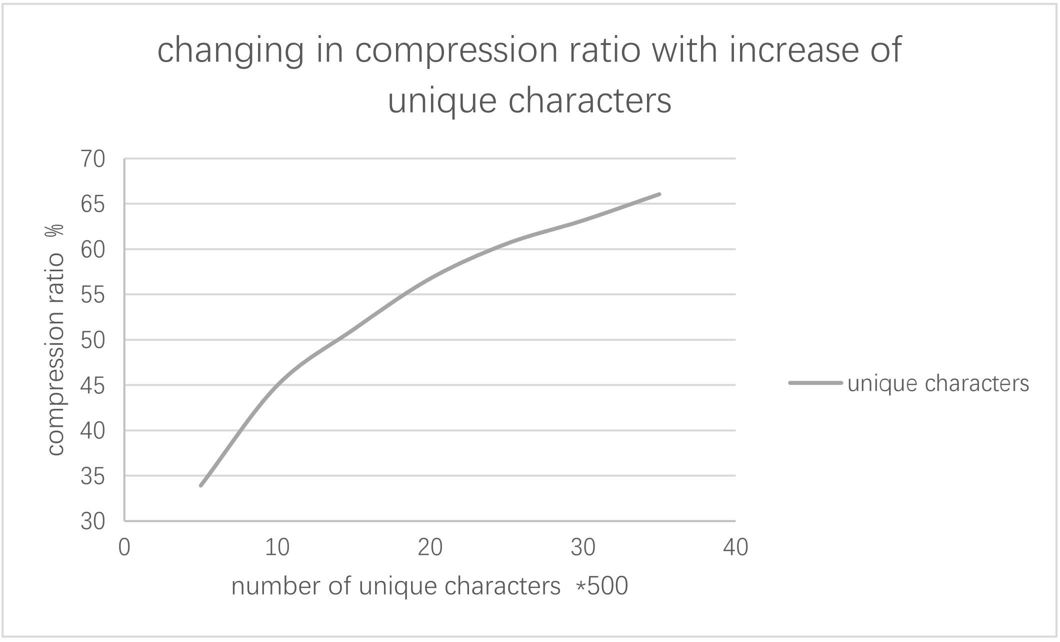
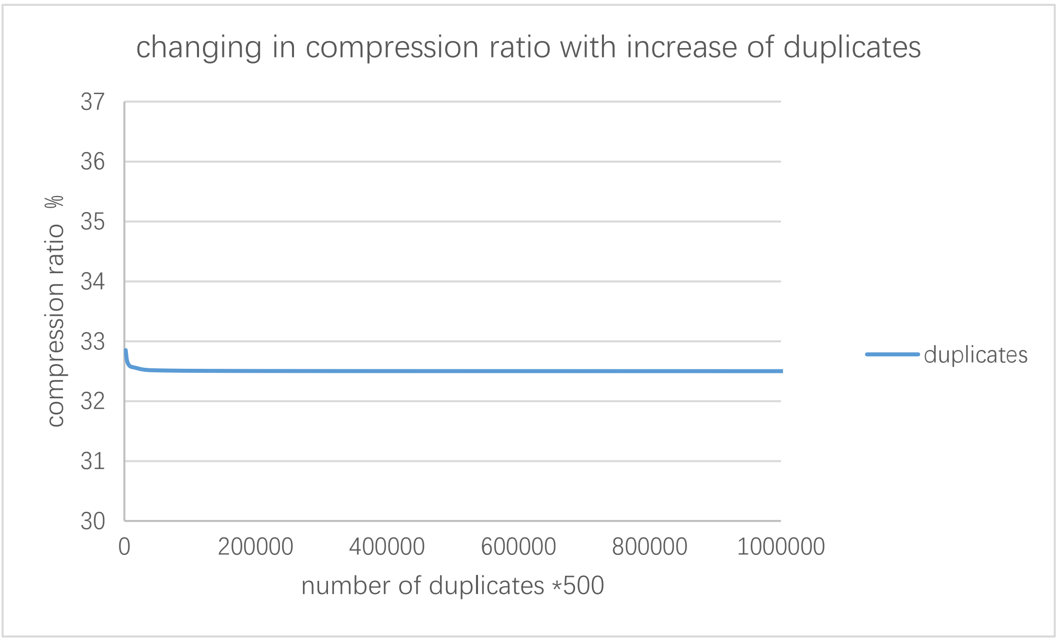
Haoran Chen

U1060286

Assignment 12 Analysis

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? Carefully describe your experiment, so that anyone reading this document could replicate your results. Submit any code required to conduct your experiment with the rest of your program and make sure that the code is well-commented. Plot the results of your experiment. Since the organization of your plot(s) is not specified here, the labels and titles of your plots(s), as well as, your interpretation of the plots is critical.**





For compressing, I expect a lower compression ratio gives a better quality of compression. In this experiment, there are two variables influencing the ratio, the frequency(of duplicates) and the number of different unique characters. Then I decided to make two experiments in order to control the variables.

The first experiment is to fix the duplicates to 500 and increase the number of unique characters. The reason to set the duplicates to 500 is I need a big enough file. When doing previous tests, I found the compressed file larger than the original file because of the header when the original file is very small. I first added 500 'a's, 500 'b's, 500 'c's, 500 'd's and 500 'e's in the txt file, then add until 'j', then 'o'… when reaching 'z', then add '0', '1', '2'… Totally I stepped up by 5 characters until 35 unique characters with 500 duplicates in the file.

The second experiments is to fix the numbers of unique characters and increase the frequency of duplicates. For the first test, I added 2^11 times of 'a' 'b' 'c' 'd' 'e's into the file, then add 2^12… I stepped up with increasing the power of 2 until 2^20, which is around 1 million.

The ratio is calculated by checking the file property and getting the file size of original file and compressed file, then multiply 100 to get a percentage.

The first graph shows that with an increase of numbers of unique characters, the ratio of compression increases. The bit code of the last added characters are larger than the very first added ones, then the size of the increased compressed files increase in a larger amount. Thus, the compression ratio increases because the size of original file increases in a fixed ratio. The efficiency of compression decreases.

The second graph shows that with a fixed number of unique characters, when duplicate increases, the ratio does not change much. It is because when increasing the frequency, the change in file size is the same of change in frequency of duplicates. As there are no new unique characters added, the tree does not change, then for adding the same characters, the size always increases in a constant ratio against the increase of the duplicate number. Thus the compression ratio does not change much when increasing the duplicates.

2. **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?**

   For the input files with one unique character with all duplicates will result a significantly reduced number of bits because for just one unique character, it will have a very short compressed code as it is the most frequently used one. In this case, even we have a very large file, we can still have a very small compressed file. For example, if we have 100,000 ‘a’s in the original file, the number of bits will be 800,000. However, if we assume the compressed code for ‘a’ here is 1, then there will only be 100,000 bits for compressed file, which is only 12.5% of original one. On the other hand, if we have a file with all unique characters and no duplicates, all the weights of characters are 1, then there will be very small compression when compressing the file. In this case, for example, the compressed file of a file contains 100,000 ‘a’s could still be smaller than a compressed file of a file contains 5000 different unique characters.

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

   Because when we merge two trees, we add a parent node above those two nodes. Then the level of the two nodes are lower. In this case, the ones we merge first will be at the bottom of the tree. For this algorithm, we need the most frequent appearing character to be at top, which will give the shortest bit code. Therefore we need to guarantee that we merge the smallest-weight trees first to keep them at the bottom and we merge the highest-weight character at last to keep it at the top of the tree.

4. **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).**

   Huffman’s algorithm perform lossless data compression. When we decompress the compressed file, all the original data remains without any change, which fixes the definition of lossless. An example of lossy compression is to compressing a flac file into an mp3 file. When compressing, it will delete some frequency which human beings are difficult to hear. In this case if we want to decompress it, it will never get the original flac file as some data has already lost.

5. **How many hours did you spend on this assignment?**

   About 5 hours.