4/30/2020

COMP20290 Algorithms

Huffman Compression

Never Follow The Concept Of Greedy Algorithm Because Life Doesn't Work That Way!!!

Nivetha Natarajan

Ronit Dahiya
UNIVERSITY COLLEGE DUBLIN

TABLE OF CONTENTS

INTRODUCTION

In this assignment we will be building our own utility for compressing text which we can also use to compress our own files. For this assignment we will implement Huffman coding suite equipped with both compression and decompression of text files.

Huffman compression is a lossless compression algorithm which can be used with files of any type. It help us to reduce the storage of files by upto half in some situation. This algorithm follows **greedy algorithm.**

Let's begin!

TASK 1 - HUFFMAN TREE OF PHRASE BY HAND

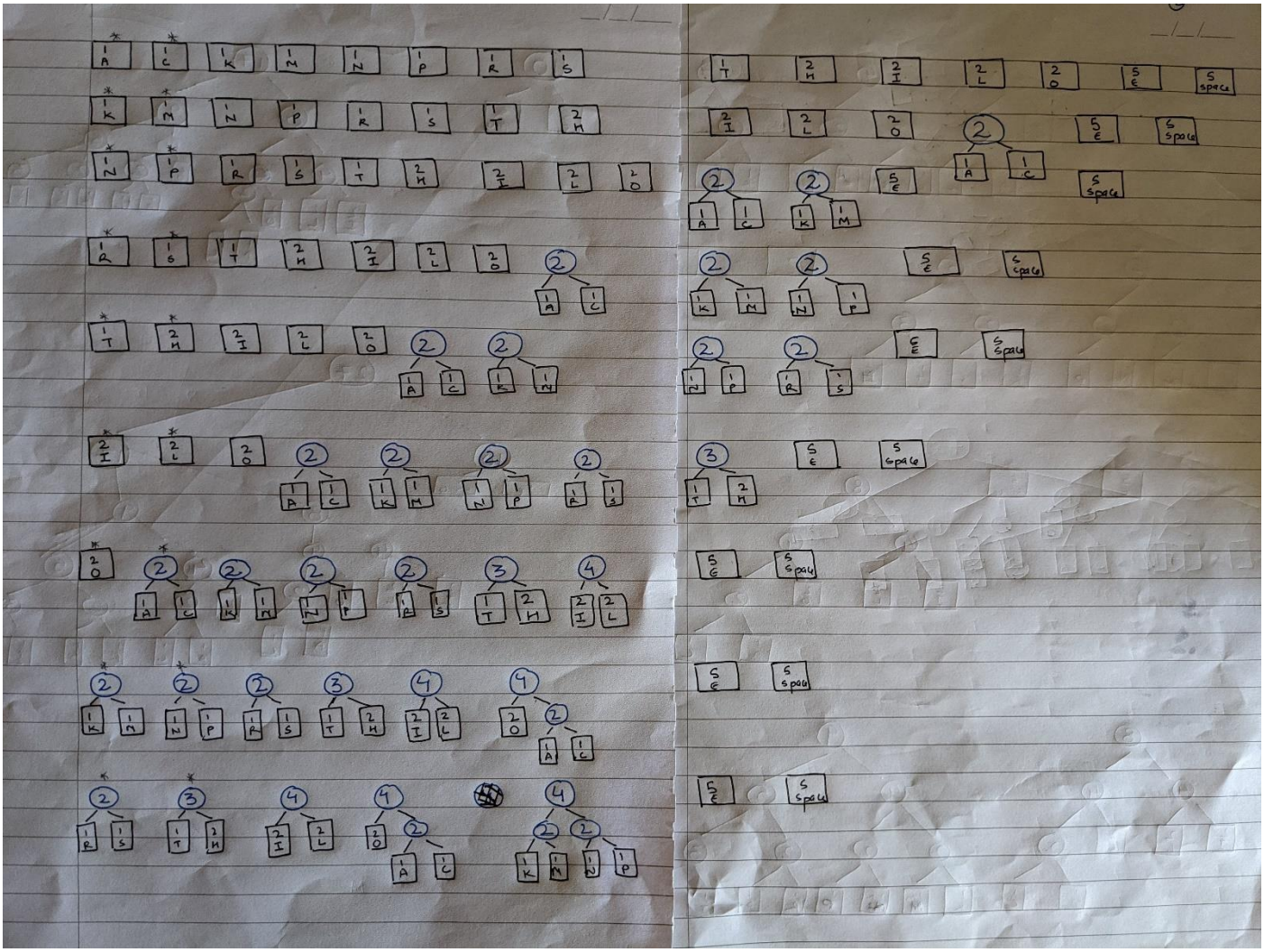
There is no place like home

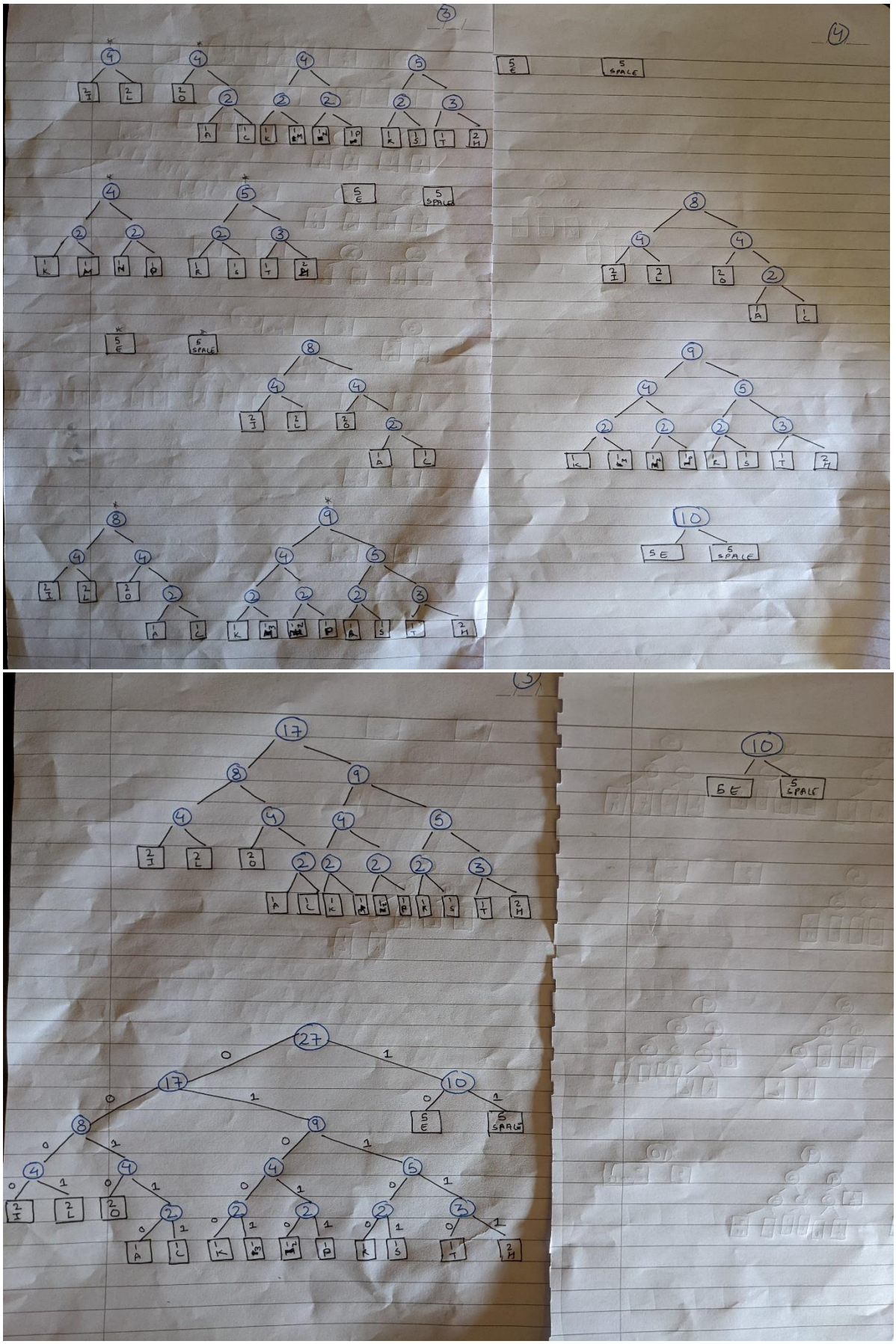
- First, we will count individual character in the given phrase. Total Digit Count: **27** Number of Character/Key: **15**

Key	Count
A	1
C	1
E	5
H	2
I	2
K	1
L	2
M	1
N	1
O	2
P	1
R	1
S	1
Space	5
T	1

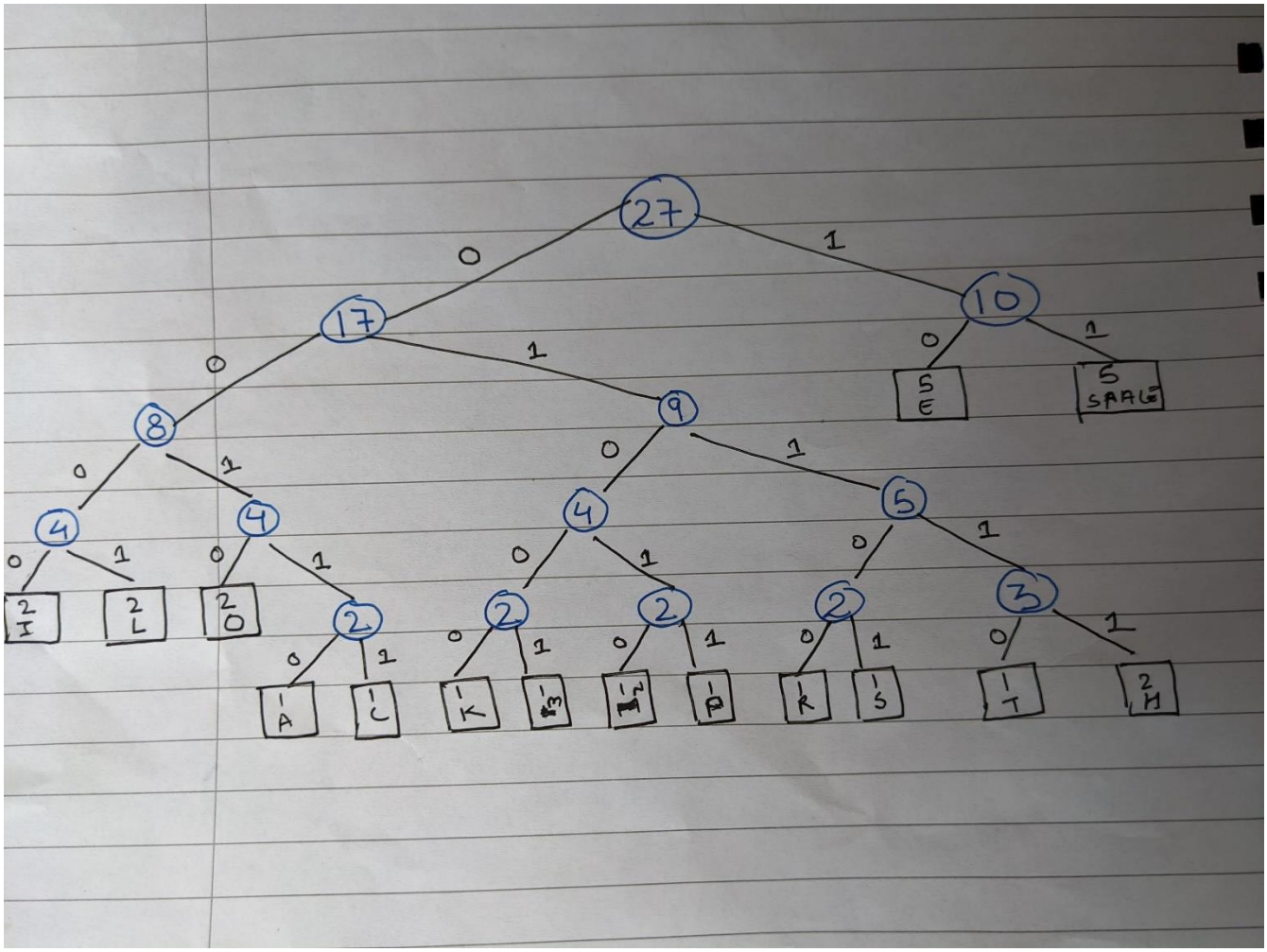
The **Huffman tree** is as follow:

- First, create a collection of **n** initial Huffman trees, each of which is a single leaf node containing one of the letters.
- Put the **n** partial trees onto a priority queue organized by weight (frequency).
- Next, remove the first two trees (the ones with lowest weight) from the priority queue.
- Join these two trees together to create a new tree whose root has the two trees as children, and whose weight is the sum of the weights of the two trees. Put this new tree back into the priority queue.
- This process is repeated until all the partial Huffman trees have been combined into one.





- Now, Beginning from the root, we assign either a '0' or a '1' to each edge in the tree.
- **'0'** is assigned to edges connecting a node with its left child, and
- **'1'** to edges connecting a node with its right child.



CODE WORD TABLE

Key	Values
A	00110
C	00111
E	10
H	01111
I	0000
K	01000
L	0001
M	01001
N	01010
O	0010
P	01011
R	01100
S	01101
Space	11
T	01110

TASK 2 – BUILDING YOUR OWN HUFFMAN COMPRESSION SUITE

In this task, I have implemented Huffman compression which can read in and give output files. I can compress as well as decompress the files. An encoding Trie is build, written as trie as upstream (for decompression) and used this trie to encode the input byte-stream as bitstream. For decompression, trie is encoded and used to decompress the bitstream.

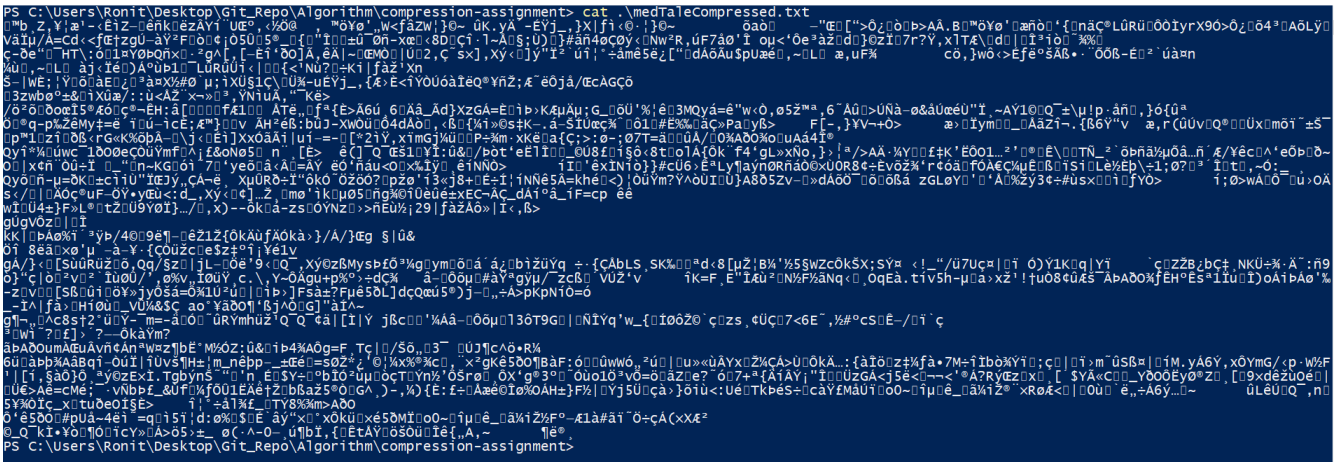
After the program is build successfully, command is given in CMD in the Directory telling our function to compress or decompress as shown below:

```
C:\Users\Ronit\Desktop\Git_Repo>cd Algorithm

C:\Users\Ronit\Desktop\Git_Repo\Algorithm>cd compression-assignment

C:\Users\Ronit\Desktop\Git_Repo\Algorithm\compression-assignment>java Huffman compress < data/medTale.txt > medTaleCompressed.txt
```

- **Command**: **java Huffman compress < data/medTale.txt > medTaleCompressed.txt**
- **With this command medTale.txt is compressed and stored in a new file named as medTaleCompressed.txt**



- **Command**: **cat .\medTaleCompressed.txt (Command is given in PowerShell in same directory)**
- **With this command we preview the compressed file i.e. medTaleCompressed.txt**

```
C:\Users\Ronit\Desktop\Git_Repo\Algorithm\compression-assignment>java Huffman decompress < medTaleCompressed.txt
it was the best of times it was the worst of times
it was the age of wisdom it was the age of foolishness
it was the epoch of belief it was the epoch of incredulity
it was the season of light it was the season of darkness
it was the spring of hope it was the winter of despair
we had everything before us we had nothing before us
we were all going direct to heaven we were all going direct
the other wayin short the period was so far like the present
period that some of its noisiest authorities insisted on its
being received for good or for evil in the superlative degree
of comparison only
```

```
C:\Users\Ronit\Desktop\Git_Repo\Algorithm\compression-assignment>java Huffman decompress < medTaleCompressed.txt  > medTaleUncompressed.txt
```

- **Command**: **java Huffman decompress < medTaleCompressed.txt > medTaleUncompressed.txt**
- **With this above command, file is Decompressed.**
- **With the second command, Decompressed file is stored in new txt file named as medTaleDecompressed.txt**

THE SAME STEPS ARE REPEATED FOR OTHER FILES TOO. EXACT SAME CODE WITH OTHER FILES

The above txt/bin compressed, and decompressed file is pushed in the repo along with the code.

TASK 3 – COMPRESSION ANALYSIS

In this task we aim to test our algorithm and comparing it with other.

File name	Normal file size	Compressed file size	Uncompressed file size	Compression Ratio (Compressed/original)	Compression Time(s)	Decompression Time(s)
genomeVirus.txt	50008	12576	50008	0.251479763	13.831445	8.244611
medTale.txt	45056	23912	45056	0.53071733	20.285754	16.095801
mobyDick.txt	9531704	5341208	9531704	0.560362344	230.567936	124.897082
q32x48.bin	1536	816	1536	0.53125	11.546762	7.721013
TerminatorStory	54304	32320	54304	0.595167943	16.4408	7.6295

Compression Time **(genomevirus.txt): 8.2446**

Compression Ratio: **0.2514**

Compression Time **(medTale.txt): 16.095**

Compression Ratio: **0.530**

Compression Time **(mobydick.txt): 124.8970**

Compression Ratio: **0.560**

Compression Time **(q32x48.bin): 7.721**

Compression Ratio: **0.53125**

Compression Time **(TerminatorStory.txt): 16.4408**

Compression Ratio: **0.5951**

ANALYSIS

We observed that as the file size increase, the computation time also increase. Size directly proportional to time.

This algorithm was although able to compress the file by approx. 46-50%. Which is quite significant. It reduces the size by half. We can also conclude that Huffman is more efficient than Runlength for compression.

Q3. WHAT HAPPENS IF YOU TRY TO COMPRESS ONE OF THE ALREADY COMPRESSED FILES? WHY DO YOU THINK THIS OCCURS?

When we compress the already compressed file, the size of file increases. This happens because of Overhead.

We perform compression in order to remove the redundancies. So, with compression performed once, we removed these redundancies. But now when we compress again, it doesn't compress further (no redundancies left) and in turn it increase the size of overhead. This increase the size of the file.

For example, I compressed **genomevirusCompressed -> genomevirusCompressionPerformedTwice.txt**. The files sizes is increased from 12576 to 14896 bits. Thus, verifying our theory.

Q4. USE THE PROVIDED RUNLENGTH FUNCTION TO COMPRESS THE BITMAP FILE Q32X48.BIN. DO THE SAME WITH YOUR HUFFMAN ALGORITHM. COMPARE YOUR RESULTS. WHAT REASON CAN YOU GIVE FOR THE DIFFERENCE IN COMPRESSION RATES?

The q32x48.bin file is compressed from **1536 bits** (**192 bytes) to 816 bits (102 bytes)** using the **Huffman compression**. When we perform the same compression using the **Runlength encoding**, we reduce/compress the file

Size from **1536 bits (192 bytes) to 1144 bits (143 bytes)**.

This means the compression performed using Huffman is more efficient than the RunLength encoding. This is because Huffman coding uses the greedy approach. i.e. each step the algorithm chooses the best available option. It build a binary tree from bottom up.

Whereas, RunLength encoding, in the worst case generates output data twice more than the size of input data. This happens because of fewer amount of runs in source file. Therefore, the algorithm does not provide significant improvement over the original file.