Simon Lee
CSE 13S

WRITEUP

<u>Introduction:</u>
Hamming codes which is what we are instructed to make for this assignment is an error correction algorithm where we generate a nibble of parity bits constructing an (8,4) hamming code. The first 4 bits being the message and the next 4 being the parity bits. Then we have to use this newly generated hamming code to find the error syndrome. And if the error syndrome is one in which we can correct, we will know which index specifically was flipped and flip it back. However it is important to note that there are cases where the hamming code is unfixable. In order to find the hamming codes, and the error syndrome there is a systematic method that takes place in which you have to multiply a Generator matrix and a H transpose matrix which lists out all the error syndromes per row. Now that we know what is going on in the program we should talk about the error.c file

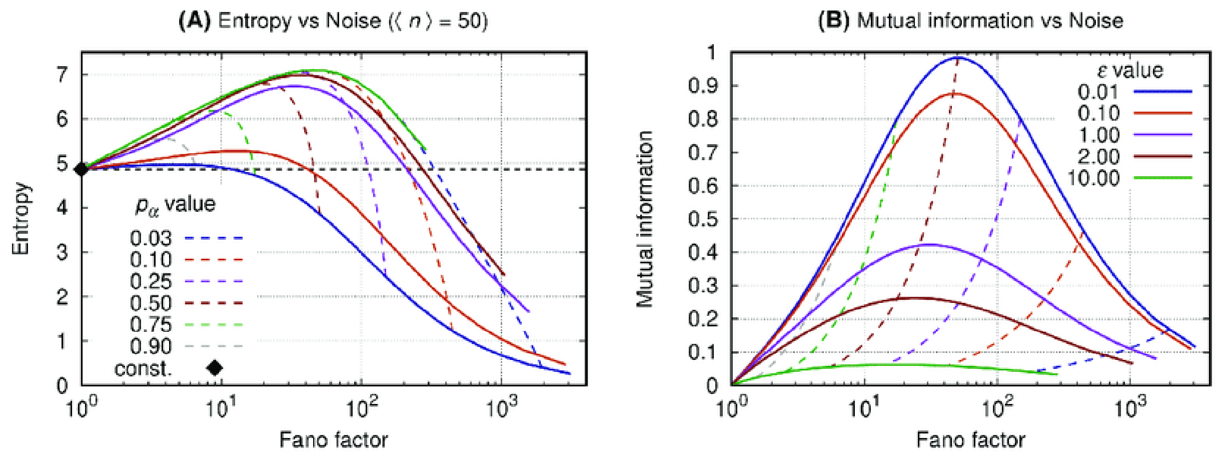
<u>Error.c</u>
Error.c was not a file we had to make but still important regardless. Since the programs encode.c and decode.c cannot actually simulate errors occurring, we need to manually add an error program that will inject "noise" or errors into the program so some bits will be flipped. This introduces another program/concept that we will soon talk about known as entropy and this write up will entail why the entropy is very high for some files and not for others.

<u>Entropy.c</u>
According to Google's Dictionary entropy is, "(in information theory) a logarithmic measure of the rate of transfer of information in a particular message or language." Another definition which I think does justice is " a lack of order or predictability; gradual decline into disorder." I think both these definitions will be of great use to understand what exactly entropy is doing and what it means. Entropy.c returns a metric which will be some random logarithmic number which will have high entropy the more random the characters are.

<u>*example on entropy*</u>:
Let's say we have some arbitrary .txt file with the contents 'aaaaaaaaaaaaaaaaaaaaaaaaa'.
This file at the moment contains 25 a's. And now lets run the hamming code generator.
After finding the encoded file we get something like 'aaaabaaaaaaaaaaaaaaaaaaaa'.
We see that there is now a random b in the file in which a bit flip occurred thanks to the error.c file.
We can easily detect that the b is in the wrong place and must be flipped back to a. Since there were only the letters 'a' to begin in the file this would be considered to have<u> low entropy.</u>

__Additional example on entropy:__
Let's now say we have another arbitrary .txt file with the contents ‘abcdefghijklmnopqrstuvwxyz’
This file now contains every letter once for a total of 26 characters. And after running the encoder.c plus the error.c we get ‘abcdeeghijklmnopqrstuvwxyz’ .
I have highlighted where the error has occurred in which the letter f got flipped to an e.
Although we can easily detect that an f got flipped to an e, the computer is not smart enough to detect this pattern without feeding it information. So it will go through the whole code and fix it if it has a valid error syndrome. Therefore this .txt file has __higher entropy__ than the previous example.

__Special case for entropy: error rate 1.0:__



Source:
https://www.researchgate.net/figure/Entropy-or-mutual-information-and-Fano-factor-were-obtained-varying-N-for-fixed-p-a-and_fig2_340878908

I have found this graph on the internet by the *research gate* group that can help us visualize some special cases for entropy when noise or “Error” is injected. If we look at what is defined as graph(A) they have an entropy vs noise graph which is some form of a bell curve. I have this whole section to explain why if you inject more error to the file the entropy will actually begin to decrease. So let's say we do inject an error rate of 1.0000 meaning every single bit in the file is an error. If every single bit was flipped because of “noise” we do not even know what the original file’s messages were. But if we injected 0.0020 of an error rate, entropy would be pretty high up there obviously pertaining to the file but it will know that some bits are at least correctable. So the more diverse the files content characters are, and at a reasonable amount of noise injected you can find the maximum entropy. I will be experimenting with my code to try to find the highest entropy in a later section. So what we should take from this section is that if you inject too much noise the overall entropy will decrease once it has passed the maximum. And although it is not your typical statistical bell curve, it resembles something pretty close to what it should be.

Personal Experiments to understand entropy further

One experiment that I was messing around with that was referenced earlier was to try to find the maximum entropy for the given .txt files given to us in our resources repository. These files are as follows:

a.txt - consisting of a single 'a' in a file
Aaa.txt - large amount of the letter 'a' in a file
Alphabet.txt - contains the alphabet printed many times
Random.txt - random sequence of alphabets both upper and lower case, and numbers

My hypothesis coming in was the way I listed the files from top to bottom. I think that a.txt will have the least amount of entropy at a given rate and random.txt will have the most entropy at a given rate. Refer to the table below to see what I found as results.

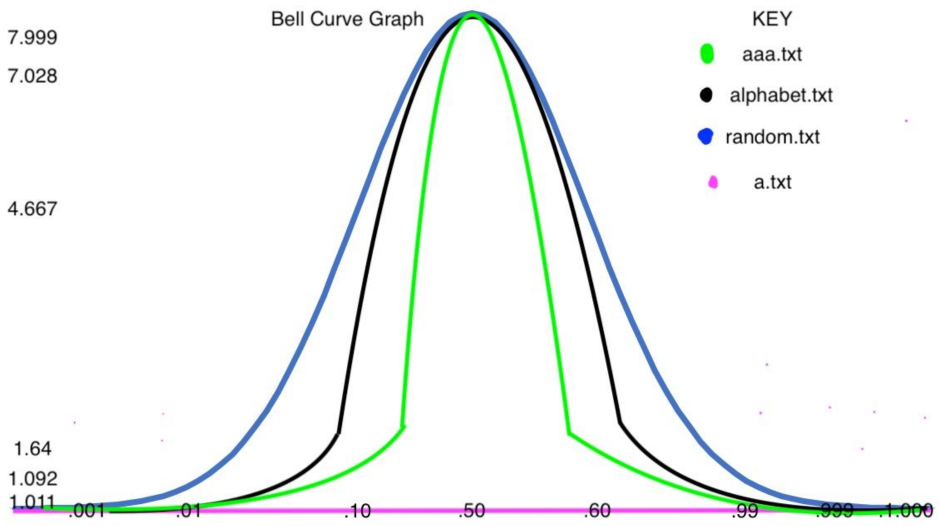
ENTROPY TABLE

File name	E = 0.0001	E=0.001	E = 0.01	E = 0.1	E = 0.25	E = 0.5	E=0.9999
a.txt	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000
aaa.txt	1.011179	1.092364	1.641210	4.667286	7.028131	7.999150	1.010970
alphabet.txt	3.141919	3.223010	3.768161	6.412802	7.696790	7.999068	3.141728
random.txt	3.545095	3.626258	4.170515	6.754802	7.836461	7.999158	3.544876

E = error rate

Analysis of Data

After running some tests on my entropy data, as expected the highest entropy occurs at an Error rate of E = 0.5. This is true because as previously stated we have a bell curve, And to my surprise aaa.txt actually had a higher max entropy than alphabet.txt. But as expected random.txt with its random sequence of capital, and lowercase letters, had the highest overall entropy. Another thing I found interesting was that aaa.txt had the largest exponential growth of the bunch. I do not know if this data is actually accurate but at the same time it makes sense. If there are only two characters in a file, let's say for example "a'' and "b", there is a good chance it can be either a or b so the entropy would be high because it has a high chance of being one of two options which gives it a bigger chance of error if its wrong. It also makes sense that alphabet.txt and random.txt starts at a higher entropy level due to how random it is from the start. I have drawn a very rough graph below which sort of resembles the relationship between the data above.



Although they are normal graphs what is evident is that the **standard deviation** of the aaa.txt is far different from the alphabet.txt and the random.txt, because the width and the exponential growth of the height is different. Width in statistics is determined by standard deviation and we can see even through a rough sketch of the graph that standard deviation is far different from the aaa.txt and the two other random ones. This all means that the overall standard deviation is less than those two random texts because the larger the standard deviation the more flat the graph would be. So we can sort of see that the randomness of random.txt and alphabet.txt share a far closer standard deviation. Please refer to the table above which will show the bell curve behavior. Though my test doesn't resemble the full bell curve, by knowing the first half of the curve, we can see what to expect of the right half of the curve. So both the graph and the table is used to sort of have real time data with a visual representation although the graph is a rough sketch.

Conclusion:

Overall I learned a lot about entropy, how error syndromes affect entropy and how the process of hamming codes can be fixed with an (8,4) hamming code. I really enjoyed comparing the data of the 4 texts that were supplied and interpreting their entropy levels. I also was able to analyze standard deviation of the overall data which explains why the widths were different. Because aaa.txt had such a low standard deviation it ended up being squished which affected the exponential growth the closer it got towards the center of the bell curve.

Apologies for the graph

I know the graph I have drawn is not the standard to what is expected. I will say this has been the hardest programming assignment that I have ever faced. Though I understood it from front to back conceptually, I really struggled to make the overall program and did the best I could with the writeup. I have started very early on this assignment and still found myself barely making it right before the deadline. I once again apologize for not displaying a better graph but will do better in the future. Thanks for taking the time to read this write up and I hope my actual analysis of standard deviation and the tests I ran were sufficient enough to make up for my lackluster graph. I really enjoyed this assignment though it was ridiculously hard for me. - Simon Lee