Comparison of Coding Methods Using MATLAB

Done By:

Aml Tarek

Mohammad Mahmoud

Youssef Allam

Part 1: Uniform Discrete Distribution

For this part we have M symbols where we will test for M=4,6,8 and we will compare the fixed length coding to the Huffman coding and to the entropy. The results are summarized in the table below. The code was tested over a string of length N=20 generated by a sequence of numbers from 1 to M.

M	Fixed Coding	Huffman Coding	Entropy
4	**Total Length**	**Total Length**	2
	40	40	
	Average Length	**Average Length**	
	2	2	
6	**Total Length**	**Total Length**	2.58
	60	52	
	Average Length	**Average Length**	
	3	2.67	
8	**Total Length**	**Total Length**	3
	60	60	
	Average Length	**Average Length**	
	3	3	

By looking at the above results, we see that the key benefit from Huffman coding comes from the case where the number of symbols is not an exact power of two. The fixed length coding uses $\lceil log_2 M \rceil$. When M is an exact power of two the ceil has no effect and the fixed length coding is already optimal as it is equal to the entropy. However, when M is not a power of 2 (such as M=6) the fixed length coding is not optimal as the Huffman coding provides an average length closer to the entropy.

Part 2: Arbitrary Distributions

Next we perform the same process we did in part 1 however we do it over the arbitrary distributions defined in the problem description. The results are as follows:

Symbol	Fixed Coding	Huffman Coding	Entropy
Y	**Total Length**	**Total Length**	1.94
	60	59	
	Average Length	**Average Length**	
	3	1.94	
Z	**Total Length**	**Total Length**	2.39
	60	54	
	Average Length	**Average Length**	
	3	2.45	

For dataset Y, the symbol distribution is highly non-uniform, resulting in a much lower entropy (1.94 bits/symbol) than the fixed-length requirement (3 bits/symbol). Huffman coding approaches the entropy limit with an average length of 1.94 bits/symbol, achieving strong compression relative to fixed coding.

For dataset Z, the distribution is less skewed, meaning the gap between entropy (2.39 bits/symbol) and fixed length (3 bits/symbol) is smaller. Huffman coding improves the average code length to 2.45 bits/symbol, giving moderate compression.

The raw number of bits returned by huffmanenco includes padding/overhead, so the correct performance comparison is based on the reported average bit lengths, not the raw total bit count.

Part 3: Huffman Encoding of Text File

This part uses Huffman encoding to encode a sample text file. It returns the compression rate (Compared to the ascii encoding) and compares the binary size of both.

--- Huffman Compression Report ---

Cleaned input length (chars): 6207

ASCII Size (bits):   49656

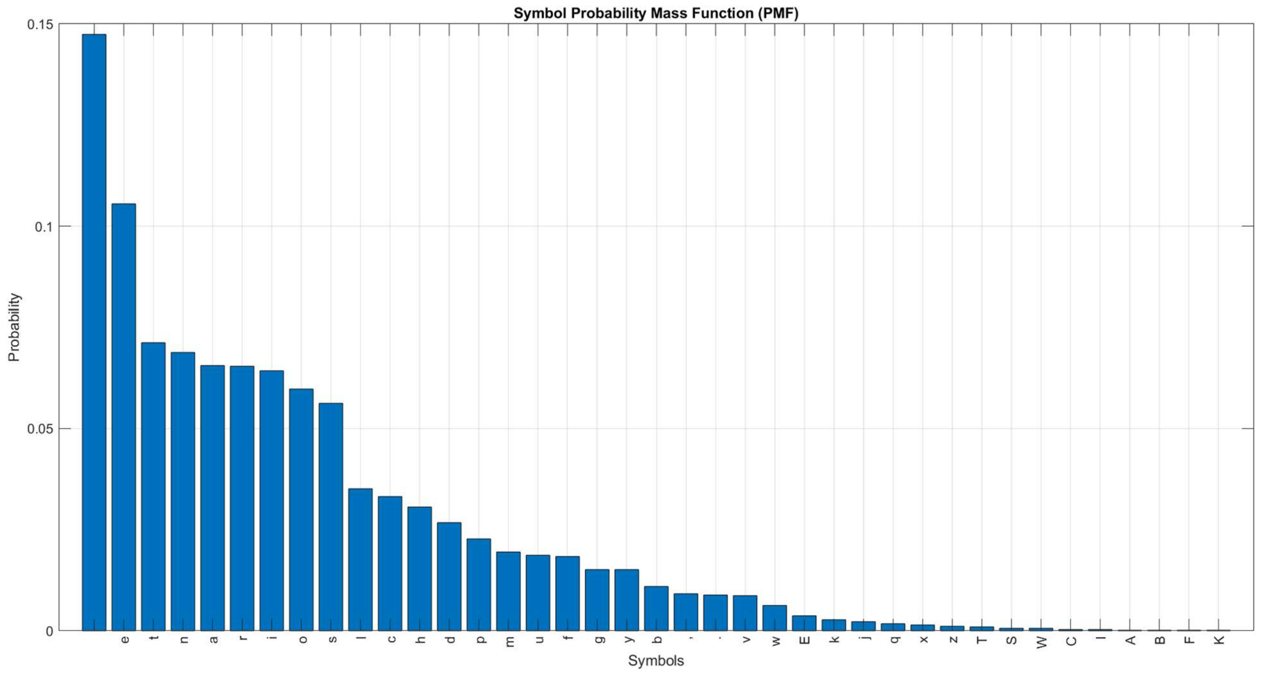
Huffman Size (bits):   26436

Compression (vs ASCII):   46.76%

Decoded text saved: Sample Text (2)_huffman_decoded.txt

Binary files saved: Sample Text (2)_huffman.bin, Sample Text (2)_ascii.bin

This shows how efficient the Huffman encoding could be and we can expect from this result that the PMF of the text document is highly skewed which agrees with the PMF generated using MATLAB shown below



Part 4: Fano Encoding

Similarly, we encode the data using Fano encoding and get the below results.

```
--- Fano Compression Report ---

Cleaned input length (chars): 6207

ASCII Size (bits):        49656

Fano Size (bits):         27662

Compression (vs ASCII):     44.29%

Decoded text saved:        Sample Text (2)_fano_decoded.txt

Binary files saved:        Sample Text (2)_fano.bin, Sample Text (2)_ascii.bin
```

The above results show that Fano has lower compression than Huffman which is consistent with the well-known fact that Huffman is either as efficient or more efficient than Fano. The main advantage of Fano is its ease of implementation and low complexity compared to Huffman.