CS 527 Fall 2014
Term Paper

Spencer Hubbard
Oregon State University
Corvallis, OR 97331
hubbarsp@onid.oregonstate.edu

William Leslie
Oregon State University
Corvallis, OR 97331
lesliew@onid.oregonstate.edu

Francis Vo
Oregon State University
Corvallis, OR 97331
vof@onid.oregonstate.edu

1 Introduction

For each $i \in \{1, \ldots, n\}$, let x_i be a symbol and let p_i be the probability associated with x_i. Let $f_i = \sum_{j=1}^{i-1} p_j + p_i/2$ and let $\ell_i = \lceil \log 1/p_i \rceil + 1$. Now, let c_i be the first ℓ_i bits in the binary expansion of the number q_i. Shannon-Fano-Elias code uses c_i as the codeword for x_i. It can be shown that the length of the codewords satisfy Kraft's inequality, i.e., $\sum_{i=1}^{n} 2^{-\ell_i} \leq 1$, which means that the code is uniquely decodable. It can also be shown that the code is a prefix code. Note that we do not assume that the symbols are indexed by decreasing probability. When the symbols are indexed by decreasing probability, this is just Shannon-Fano code.

Suppose that Alice wants to send a secret message to Bob but their adversary Eve—the eavesdropper—is allowed to intercept and attempt to read any message that Alice sends Bob. Suppose also that Alice, Bob, and Eve all know the set of symbols and their associated probabilities. Notice that for a fixed set of n symbols and associated probabilities, there are $n!$ Shannon-Fano-Elias codes corresponding to the $n!$ permutations of the indices $\{1, \ldots, n\}$. This means if Alice and Bob can agree upon a particular permutation without Eve knowing, then Alice can use Shannon-Fano-Elias code to send a message to Bob and Eve must potentially check $n!$ permutations before successfully reading the message.

2 Huffman Code vs Shannon Code

Huffman code is an optimal prefix code and could be used for data compression, but it cannot be used for encryption. Since the probability of each symbol is very important the encryption, knowing fixed set of n symbols, associated probabilities, and the algorithm used will easily yield the correct encoding. Shannon-Fano-Elias Coding is extremely sub-optimal, but great for encryption. Even with the knowledge of the fixed set of n symbols, associated probabilities, and the algorithm used, there are still $n!$ different encodings. Shannon-Fano-Elias Coding depends greatly on the order of the the symbols as well as the probabilities.

3 Improved Shannon-Fano-Elias Coding

3.1 "Better" PMF for encoding

Since the symbols are not necessarily indexed by decreasing probability, it follows that the resulting code may have a large average length. To decrease the average length of the code, we will replace the true probabilities with some temporary probabilities and construct the code using these temporary probabilities.

For each $i \in \{1, \ldots, n\}$, let x_i and p_i be defined as before. Let ℓ_i be the length of the Huffman codeword for the symbol x_i with probability p_i and let L be the average length of this Huffman code, i.e., let $L = \sum_{i=1}^{n} p_i \ell_i$. Now, let $p'_i = 2^{-\ell_i}$ be the temporary probability of the symbol x_i and let ℓ'_i be the length of the Shannon-Fano-Elias codeword for the symbol x_i with the temporary probability p'_i, i.e., let $\ell'_i = \lceil \log_2 1/p'_i \rceil + 1$. Also, let L' be the average length of this Shannon-Fano-Elias code with respect to the true probabilities, i.e., let $L' = \sum_{i=1}^{n} p_i \ell'_i$. Notice that $\ell'_i = \ell_i + 1$. This means

$$\begin{aligned} L' &= \sum_{i=1}^{n} p_i \ell'_i \\ &= \sum_{i=1}^{n} p_i \left(\ell_i + 1\right) \\ &= \sum_{i=1}^{n} p_i \ell_i + \sum_{i=1}^{n} p_i \\ &= L + 1 \end{aligned}$$

which means the average length of this Shannon-Fano-Elias code is within one of optimal.

Using the the Shannon-Fano-Elias Code in Table **??**, we will generate a better PMF. First we have to generate a Huffman code of the symbols. We find out that each codeword should have length 2 and therefore each $p'_i = 0.25$. We will then use p'_i, to generate a new Shannon-Fano-Elias code.

Symbols	p_i	f_i	Codeword
D	1/4	1/8	001
C	1/6	1/3	0101
B	1/4	13/24	100
A	1/3	5/6	110

Table 1: Base Shannon-Fano-Elias Code

3.2 Codeword Truncation

In Figure **??**, each codeword has the rightmost bits are truncated while it and its preceding and succeeding codewords are mutually prefix-free. Since the algorithm removes bits from the codeword, it will reduce the average code length and therefore optimize the code. This works because the closer

Symbols	p_i'	f_i'	Codeword
D	1/4	1/8	001
C	1/4	3/8	011
B	1/4	5/8	101
A	1/4	7/8	111

Table 2: Shannon-Fano-Elias Code using better PMF

codes are, the more similar they are due to the codeword being based on the running probability. Therefore, the codeword does not have to be compared to every other codeword and only the ones immediately before and after it.

Figure 1: Figure text here.

```
1 Input: Shannon-Fano-Elias codeword set codeword(xi) for 1<=i<=n.
2 Output: codeword(xi) with lower expected length.
3     for 1<=i<=n do
4         c(xi)<- codeword(xi)
5         while codeword(xi-1),c(xi) and codeword(xi+1) are mutually
              prefix-free do
6             codeword(xi) <- c(xi)
7             c(xi) <- removing the rightmost bit of the codeword(xi)
8         end while
9     end for
```

Using the algorithm in order on Table **??**, we have 001 compared to 0101. These codewords are mutually prefix-free and the rightmost bit and be deleted. D is now 00. Removing one more would result in 0, which is a prefix of 0101 and the algorithm continues to the next codeword.

0101 is compared to 00 and 100. These codewords are prefix-free and the rightmost bit can be deleted. C is now 010. 010 is compared to 00 and 100. These codewords are still prefix-free and the rightmost bit can be deleted. C is now 01. Deleting one more would result and 0 and therefore a prefix of 00, thus the algorithm can move to the next codeword.

100 is compared to 01 and 110. These codewords are prefix-free and the rightmost bit can be deleted. B is now 10. Deleting one more bit would result in 1, which is a prefix of 110 and therefore the algorithm moves on.

110 is compared to 10. These codewords are prefix-free and the rightmost bit can be deleted. A is now 11. Deleting one more bit would result in 1, which is a prefix of 10, therefore the algorithm continues. The algorithm then ends because our codeword set has been exhausted.

Our resulting code is given in table **??**. Here we see a significantly reduced code. The average length has been reduced to 2, as opposed to the 3.2 that it was previously. Not only is this code significantly reduced in this case, it is also optimal. Thus, in certain cases, this truncation method can result in an optimal Huffman code.

Symbols	p_i	f_i	Codeword
D	1/4	1/8	00
C	1/6	1/3	01
B	1/4	13/24	10
A	1/3	5/6	11

Table 3: Caption text here

4 Finding the Codeword Set Used for Encoding

Cryptanalysis is very hard for Shannon-Fano-Elias Code because there are $n!$ different encodings. Decryption uses a $O(n^3)$ algorithm to check if the codeword set could have been used to generate the sequence. We will need to use this algorithm for all sets of codewords. There are $n!$ different sets of codewords. The algorithm will create a subset of sets of codewords that work on the sequence. Another sequence can be used to make a smaller subset.

4.1 Example 1

Given the received encoded sequence **??** and a candidate codeword set 11110,110,01. The codeword set needs to be listed in order by length.

Received encoded sequence: 0101010100100101000010000101010100100000011000001

The codeword 11110 doesn't appear in the sequence **??**, so we have to search for the codeword 110. The bold numbers represent the codeword 110 that were found, and the underlined numbers are codewords that are found before the codeword 110.

Searching for 110 in sequence 0101010100100101000010000101010100100000**110**00001

For every codeword, we will delete it from the sequence **??** if any other codewords (that we haven't checked) appear before it. But codewords 110,01 were not found before the codeword 110, so the current sequence is still unchanged from sequence **??**. Now we search for the codeword 01.

Searching for 01 in sequence: **<u>01010</u>10100<u>10</u>10**0000**0**1000**<u>01010</u>10100**1000**0**11000**01**

After searching for all occurrences of the codeword 01, we remove the occurrences that have the codewords 01 to the left of it

After removing 01 from sequence: 0010010000010000100100000011000001

Since the sequence is not empty, after we searched using on the codewords.

4.2 Example 2

Once again, we are still using sequence **??**. This time we are checking if the codeword set 00001,010,10. Fist we will search for the longest codeword in the set in the sequence.

Searching for 00001 in sequence: 0101010100100101000010000101010100100000110000 1

We found 4 occurrences of the codeword 00001, and they each have another codeword to the left of it. Therefore we will delete all of them. We will continue searching though the new sequence for the next codeword 010.

Searching for 010 in sequence after removing all 00001: **0101010100100101001010100101010**

We found 12 occurrences of the codeword 010, but not all of them had other codewords before it. So we only deleted 4 occurrences. We will search for the last codeword 10.

Searching for 10 in sequence after removing all 010: **1010101010101010**

5 Conclusion

References

[1] Rajendra Katti and Xiaoyu Ruan. Cryptanalysis of shannon-fano-elias codes. In *Information Theory, 2005. ISIT 2005. Proceedings. International Symposium on*, pages 1231–1235. IEEE, 2005.

[2] Xiaoyu Ruan and Rajendra Katti. Using improved shannon-fano-elias codes for data encryption. In *Information Theory, 2006 IEEE International Symposium on*, pages 1249–1252. IEEE, 2006.