# CS 527 Fall 2014
# Term Paper

Spencer Hubbard

## 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 $q_i = \sum_{j=1}^{i-1} p_j + p_i/2$ and let $\ell_i = \lceil \log 1/p_i \rceil + 1$. Now, let $c_i$ be the first $\ell_i$ 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 with only a $1/n!$ chance of Eve successfully reading the message.

## 2 Improved Shannon-Fano-Elias Coding

### 2.1 "Better" PMF for encoding

This improvement to Shannon-Fano-Elias code aims to change the PMF used for the encoding. Since Huffman coding is more optimial, we will use the length of each symbol to generate a new PMF.

### 2.2 Codeword Truncation

Input: Shannon-Fano-Elias codeword set codeword(xi) for 1¡=i¡=n. Output: codeword(xi) with lower expected length. for 1¡=i¡=n do c(xi)¡- codeword(xi) while codeword(xi-1),c(xi) and codeword(xi+1) are mutually prefix-free do codeword(xi) ¡- c(xi) c(xi) ¡- removing the rightmost bit of the codeword(xi) end while end for

In this algorithm, 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 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.

Below we have a code with 4 codewords:

| Values | P(X) | F(X) | 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 |

Using the algorithm in order we have 001 compared to 0101. These numbers are mutually prefix-free and the rightmost value and be deleted. D is now 00. Removing one more would result in 0, which is not prefix free and the algorithm continues to the next value.

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

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

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

Our resulting code is

| Values | P(X) | F(X) | 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 |

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.

# 3 Finding the Codeword Set Used for Encoding

# 4 Conclusion