1 A bit of History

Arithmetic coding is a lossless data compression technique developed in the seventies, whose invention is credited to Jorma Rissanen then at IBM Research Labs. Arithmetic coding is usually more practical than Huffman coding when operating on a large number \( n \) of source symbols, but because algorithms are protected by several US patents, many compression utilities do not use it. For instance, while the original bzip used an arithmetic coder, its replacement bzip2 reverted to a Huffman coder.

2 Arithmetic Coding Principle

In this section, we describe the high-level concept of arithmetic coding with an example. We consider a memoryless source \( X \in \{1, 2, 3, 4\} \) such that

\[
\begin{align*}
    p_X(1) &= 0.25, \\
    p_X(2) &= 0.5, \\
    p_X(3) &= 0.2, \\
    p_X(4) &= 0.05,
\end{align*}
\]

whose entropy is \( \mathbb{H}(X) \approx 1.68 \) bits.

The Huffman code to encode 1 symbol at a time would consist of the following four codewords

\[
\begin{align*}
    c_1 &= 01, \\
    c_2 &= 1, \\
    c_3 &= 001, \\
    c_4 &= 000,
\end{align*}
\]

whose average length is \( \approx 1.75 \) bits.

To get closer to entropy, we known that we can encode \( n > 1 \) symbols at a time. If we were to implement Huffman coding, we would need to construct the entire codebook of \( |\mathcal{X}|^n \) codewords and store it. This procedure becomes quickly prohibitive, for instance with \( |\mathcal{X}| = 4 \) and \( n = 10 \), we would need to store over 1 million values. Arithmetic coding provides a way of generating codewords for any \( n \), \textit{without precomputing all codewords}. The key idea of arithmetic coding is to use Shannon-Fano-Elias coding by computing the value of the CDF \( F_X^n(x^n) \) efficiently on the fly.

Assume that we take \( n = 4 \) and that we want to encode the sequence \( x = 2313 \). Note that if we order all source sequences of length 4 using the lexicographic order

\[
1111 < 1112 < 1113 < 1114 < 1121 \ldots
\]

then we can compute \( F_{X^4}(3221) \) as follows:

\[
F_{X^4}(2313) = \sum_{x^4 \leq 2313} p_{X^4}(2313) = F_{X^4}(224) + p_{X^4}(231) F_X(3).
\]

By reiterating this procedure, we can compute \( F_{X^4}(2313) \) recursively as

\[
F_{X^4}(2313) = F_{X^4}(224) + p_{X^4}(231) F_X(3)
= p_1 + p_2 F_X(2) + p_{X^4}(231) F_X(3)
\]

We can illustrate this procedure graphically as follows
In other words, we use the sequence 2313 to select a point in the interval \([0, 1]\) and use Shannon-Fano-Elias coding. For the specific source given above, we have

\[
F_{X^4}(2312) = 0.64375, \quad F_{X^4}(2313) = 0.64875,
\]

and the value of the midpoint is therefore \(x = 0.64625\). In addition \(p_{X^4}(2313) = 0.005\), and we need to truncate the binary expansion of \(x\) at \(\lceil \log_2 \frac{1}{p_{X^4(2313)}} \rceil + 1 = 9\) bits. The final codeword is therefore,

\[
c = 101001010
\]

In practice, there are several implementation issues. In particular, we cannot compute the boundaries of the intervals with infinite precision, and there must be some rounding in the calculations. Efficient implementations can be quite tricky...