Arithmetic Coding

Arithmetic coding is the most efficient method to code symbols according to the probability of their occurrence. The average code length corresponds exactly to the possible minimum given by information theory. Deviations which are caused by the bit-resolution of binary code trees do not exist.

In contrast to a binary Huffman code tree the arithmetic coding offers a clearly better compression rate. Its implementation is more complex on the other hand.

In arithmetic coding, a message is encoded as a real number in an interval from one to zero. Arithmetic coding typically has a better compression ratio than Huffman coding, as it produces a single symbol rather than several separate codewords.

Arithmetic coding differs from other forms of entropy encoding such as Huffman coding in that rather than separating the input into component symbols and replacing each with a code, arithmetic coding encodes the entire message into a single number, a fraction n where ($0.0 \leq n < 1.0$)

Arithmetic coding is a lossless coding technique. There are a few disadvantages of arithmetic coding. One is that the whole codeword must be received to start decoding the symbols, and if there is a corrupt bit in the codeword, the entire message could become corrupt. Another is that there is a limit to the precision of the number which can be encoded, thus limiting the number of symbols to encode within a codeword. There also exist many patents upon arithmetic coding, so the use of some of the algorithms also call upon royalty fees.

Arithmetic coding is part of the JPEG data format. Alternative to Huffman coding it will be used for final entropy coding. In spite of its less efficiency Huffman coding remains the standard due to the legal restrictions mentioned above.

Arithmetic Coding Algorithm:

The arithmetic coding algorithm works from leaves to the root in the opposite direction.

1. Start with an interval [0, 1), divided into subintervals of all possible symbols to appear within a message. Make the size of each subinterval proportional to the frequency at which it appears in the message.
2. When encoding a symbol, "zoom" into the current interval, and divide it into subintervals like in step one with the new range.
3. Repeat the process until the maximum precision of the machine is reached, or all symbols are encoded.
4. Transmit some number within the latest interval to send the codeword. The number of symbols encoded will be stated in the protocol of the image format.

Example 1:

The source of information A generates the symbols {A0, A1, A2, A3 and A4} with the corresponding probabilities {0.4, 0.3, 0.2 and 0.1}. Encoding the source symbols using Huffman encoder gives:

Source Symbol	P_i	Binary Code	Huffman
A0	0.4	00	0
A1	0.3	01	10
A2	0.2	10	110
A3	0.1	10	111
L_{avg}	**H = 1.846**	**2**	**1.9**

The Entropy of the source is

$$H = -\sum_{i=0}^{3} Pi \log_2 Pi = 1.846 \text{ bit/symbol}$$

Since we have 4 symbols ($4=2^2$), we need 2 bits at least to represent each symbol in binary (fixed-length code). Hence the average length of the binary code is

$$Lavg = \sum_{i=0}^{3} Pi\, li = 2\,(0.4 + 0.3 + 0.2 + 0.1) = 2 \text{ bit/symbol}$$

Thus the efficiency of the binary code is

$$\eta = \frac{H}{Lavg} = \frac{1.846}{2} = 92.3\%$$

The average length of the Huffman code is

$$Lavg = \sum_{i=0}^{3} Pi\, li = 0.4 * 1 + 0.3 * 2 + 0.2 * 3 + 0.1 * 3 = 1.9 \text{ bit/symbol}$$

Thus the efficiency of the Huffman code is

$$\eta = \frac{H}{Lavg} = \frac{1.846}{1.9} = 97.16\%$$

The Huffman encoder has the closest efficiency to the entropy that can be obtained using a prefix code. Higher efficiency can be yielded with the arithmetic coding.

Dividing into Intervals

On the basis of a well-known alphabet the probability of all symbols has to be determined and converted into intervals. The size of the interval depends linearly on the symbol's

probability. If this is 50% for example, then the associated sub-interval covers the half of the current interval. Usually the initial interval is [0; 1) for the encoding of the first symbol.

Source Symbol	P_i	Sub-interval
A0	0.4	[0.0;0.4)
A1	0.3	[0.4;0.7)
A2	0.2	[0.7;0.9)
A3	0.1	[0.9;1.0)

Assume that the message to be encoded is **A0A0A3A1A2**. The first symbol to be encoded is **A0**. We "zoom" into the interval corresponding to "A0", and divide up that interval into smaller subintervals like before. We now use this new interval as the basis of the next symbol encoding step.

Source Symbol	New 'A0' Interval
A0	[0.0;0.16)
A1	[0.16;0.28)
A2	[0.28;0.36)
A3	[0.36;0.4)

To encode the next character "A0", we use the "A0" interval created before, and zoom into the subinterval "A0", and use that for the next step. This produces

Source Symbol	New 'A0' Interval
A0	[0.0;0.064)
A1	[0.064;0.112)
A2	[0.112;0.144)
A3	[0.144;0.16)

To encode the next character "A3", we use the "A0" interval created before, and zoom into the subinterval "A3", and use that for the next step. This produces

Source Symbol	New 'A3' Interval
A0	[0.144;0.1504)
A1	[0.1504;0.1552)
A2	[0.1552;0.1584)
A3	[0.1584;0.16)

And lastly, the final result is

Source Symbol	New 'A0' Interval
A0	[0.1504;0.15232)
A1	[0.15232;0.15376)

A2	[0.15376;0.15472)
A3	[0.15472;0.1552)

Transmit some number within the latest interval to send the codeword. The number of symbols encoded will be stated in the protocol of the image format, so any number within [0.15376, 0.15472) will be acceptable.

Let's choose the number **0.1543**. The binary representation of this number is **0.001001111**. We need 10 bits to encode the message (9 bits and the floating point). The minimum number of bits needed to fully encode the message is

$$H*N = 1.846*5= \mathbf{9.23} \text{ bit}$$

Using Huffman code, the message is encoded to **0 0 111 10 110** which need also 10 bits. The larger is the number of symbols, the wider is the gap in efficiency.

Decoding the code is a reverse approach. Let's assume the number 0.1543 has been received at the decoder. 0.1543 lies in the interval [0; 0.4), then the first symbol of the message is A0.

Then, we "zoom" into the interval corresponding to "A0", and divide up that interval into smaller subintervals. We now use this new interval as the basis of the next symbol decoding step.

Source Symbol	New 'A0' Interval
A0	[0.0;0.16)
A1	[0.16;0.28)
A2	[0.28;0.36)
A3	[0.36;0.4)

0.1543 lies in the interval [0; 0.16), then the second symbol of the message is A0. Zoom into the subinterval "A0", and use that for the next step. This produces

Source Symbol	New 'A0' Interval
A0	[0.0;0.064)
A1	[0.064;0.112)
A2	[0.112;0.144)
A3	[0.144;0.16)

0.1543 lies in the interval [0.144; 0.16), then the third symbol of the message is A3. Zoom into the subinterval "A3", and use that for the next step. This produces

Source Symbol	New 'A3' Interval
A0	[0.144;0.1504)

A1	[0.1504;0.1552)
A2	[0.1552;0.1584)
A3	[0.1584;0.16)

0.1543 lies in the interval [0.1504; 0.1552), then the fourth symbol of the message is A1. Zoom into the subinterval "A1", and use that for the next step. This produces

Source Symbol	New 'A0' Interval
A0	[0.1504;0.15232)
A1	[0.15232;0.15376)
A2	[0.15376;0.15472)
A3	[0.15472;0.1552)

0.1543 lies in the interval [0.15376; 0.15472), then the last symbol of the message is A2. The decoder stops after this as the number of symbols encoded will be stated in the protocol of the message.

Exercise 1:

The source of information A generates the symbols {A0, A1, A2, A3 and A4} with the probabilities shown in the table below. Encode the source symbols using Arithmetic encoder and Huffman encoder. The message is **A4A1A0A3A2**

Source Symbol	P_i
A0	0.4
A1	0.4
A2	0.12
A3	0.06
A4	0.02

Compare the efficiency of both codes and comment on the results.

Exercise 2:

The source of information A generates the symbols {A0, A1, A2 and A3} with the probabilities shown in the table below. Encode the source symbols arithmetic encoder if the message is **A2A0A2A3**

Source Symbol	P_i
A0	0.5
A1	0.3
A2	0.2

Compare the efficiency of both codes and comment on the results.