Which code should we use?

We have seen several examples in which the amount of compression depends on the probabilities (or frequencies) of the symbols in the sequence. Often the encoder would like to examine the sequence before coding it and choose which coding scheme to use, such that compression is best – for example, choose the Golomb constant b, or decide which Elias code to use. The trick, though, is that this choice of code needs to be communicated to the decoder.

There is a tradeoff here. If the encoder and decoder have agreed in advance on what code will be used, then this information doesn't need to be communicated. This saves bits, but it gives the encoder less flexibility. If the encoder want to be flexible by choosing the coding scheme to fit the data and compress well, then the number of bits for coding the data will be low, but there is a cost: the encoder needs to send bits that specify what the code is.

Today we are going to look at some basic schemes for sending the code up front, as a sort of "header file". By this, I just mean some set of bits at the beginning of the file that specifies how the file is encoded.

Header files: encoding a code

The problem thus far: given a sequence of symbols from some alphabet, encode this sequence using some predetermined code. If the encoder uses a prefix code $C^*(A_i)$, then a decoder can recover the original sequence of events. The key assumption up to now is that both the encoder and the decoder know what is the set of possible symbols being coded $\{A_1, A_2, \ldots, A_N\}$ and what are the codewords $C^*(A_i)$ for these events.

It is reasonable to assume that the decoder knows what are the possible symbols to be coded. (It might be ASCII.) If two people or computers are going to communicate with each other, then there has to be *some* agreement or understanding of the domain. But it is not necessary that the encoder/decoder have agreed on the details of the code. Let's look at two scenarios in which the encoder tells the decoder what is the code, prior to sending the encoded sequence.

Scenario 1

One way that would *not* work is for the encoder to just list the codewords:

$$C^*(A_1)C^*(A_2)C^*(A_3)\ldots C^*(A_N)$$

For the example, the encoder might examine the sequence and compute the relative frequencies of the A_i, and then construct a Huffman code $C^*(A_i)$. This is desirable, since Huffman codes are optimal. The problem is that there is no way for the decoder (who receives the above sequence of bits) to know where one codeword ends and the next begins.

An alternative is to specify how many bits there are in each codeword using a prefix code C for codeword length.[1] Think of this as encoding: "the codeword $C^*(A_1)$ for symbol A_1 has λ_1^* bits and these bits are ....; the codeword $C^*(A_2)$ for A_2 has λ_2^* bits and these bits are .... etc". Then the header could be:

$$C(\lambda_1^*)\ C^*(A_1)\ C(\lambda_2^*)\ C^*(A_2)\ C(\lambda_3^*)\ C^*(A_3)\ldots C(\lambda_N^*)\ C^*(A_N)$$

[1]Note this is similar in flavor to how the Elias code worked.

Note that, for this to work, the encoder and decoder must have agreed on a prefix code C for the set of positive integers – this might be a Golomb code or Elias code, for example – and on the general scheme just described.

Scenario 2

For many types of data, certain *sequences of symbols* tend to occur commonly. For example, English text consists of words (sequences of characters) separated by blanks and punctuation marks. C source code consists of identifiers, key words, etc. These files are often encoded with ASCII. An encoder could use the scenario 1 to compress a particular ASCII sequence by variable length code instead of ASCII, namely, a code such that more frequently occuring characters have shorter codewords. But this doesn't take advantage of the fact that certain (sub)sequences arise more often than others.

Suppose the encoder partitions the file into a sequence of strings (using a given partitioning method e.g. parsing) and wishes to encode this sequence of strings. For example, suppose the file starts out:

`THE_CAR_ON_THE_LEFT_HIT_THE_CAR_ON_THE_RIGHT....`

and the encoder partitions it into:

`THE`, `_`, `CAR`, `_`, `ON`, `_`, `THE`, `_`, `LEFT`, `_`, `HIT`, `_`, `THE`, `_`, `CAR`, `_`, `ON`, `_`, `THE`, `_`, `RIGHT`. . . .

where the commas indicate the boundary of the partition. The encoder would then choose a code for the set of strings in this file,

{ `THE`, `_`, `CAR`, , `ON`, `THE`, `LEFT`, `HIT`, `RIGHT`. . . . }

based on their frequencies of these strings within the file, and encode the file by replacing the strings by their codewords.

Why do this? For any file, many of the strings appear multiple times. `THE` appears four times above and blank (`_`) appears ten times! So using short codewords for frequently occuring strings would be a way of achieving compression. (Note that the encoder might also consider how many symbols there are in each string, when deciding on the codeword for a string.)

Anyhow, let's say there are M strings. We are not counting repeats here. i.e. `THE` and blank are counted just once. This set of strings is $\{S_1, S_2, \ldots, S_M\}$. This set acts as a new alphabet. Let's suppose the encoder has chosen codewords $C^S(S_1), C^S(S_2), \ldots, C^S(S_M)$ for this new alphabet. The encoder now needs to tell the decoder what each string S_i is, and what is the codeword $C^S(S_i)$ for each string.

Assume the encoder and decoder have agreed on a code C^* for the basic alphabet $\{A_1, \ldots, A_N\}$ from which each S_i is constructed. This could be the ASCII code, or it could be a code that is communicated using scenario 1. This code is agreed upon in advance, or else sent as a header file. What next?

The encoder first needs to first specify M, the number of strings. It does so by sending the codeword $C(M)$. Then, for each of these M strings $\{S_1, \ldots, S_M\}$, the encoder specifies the following:

- $|S_i|$, which is the number of symbols A_i in string S_i,

- $C^*(S_i)$, which is the code of the symbols in the string (the code extension is used here, recall lecture 2)
$$C^*(\texttt{THE}) \equiv C^*(\texttt{T})\, C^*(\texttt{H})\, C^*(\texttt{E})$$
- $\lambda_i^S = |C^S(S_i)|$, which is the number of bits of the codeword $C^S(S_i)$
- $C^S(S_i)$, the codeword for the string S_i

Thus, header file is:

$$C(M)\, C(|S_1|)\, C^*(S_1)\, C(\lambda_1^S)\, C^S(S_1)\, C(|S_2|)\, C^*(S_2)\, C(\lambda_2^S)\, C^S(S_2)\, \ldots\, C(|S_M|)\, C^*(S_M)\, C(\lambda_M^S)\, C^S(S_M)$$

Notice that this scheme uses three codes:

- $C(i)$ is a code on the positive integers. Codeword lengths are λ_i.
- $C^*(A_j)$ is a code on the base alphabet of the data. Codeword lengths are λ_i^*.
- $C^S(A_j)$ is a code on a set of strings (chosen by the encoder). Codeword lengths are λ_i^S.

As long as the encoder and decoder agree on the order in which the various pieces of information are sent, the code $C^S()$ will be unambiguiously communicated. Although this may cost alot of bits up front in the header file, the hope is that there will be a net savings because the data can be compressed very well.