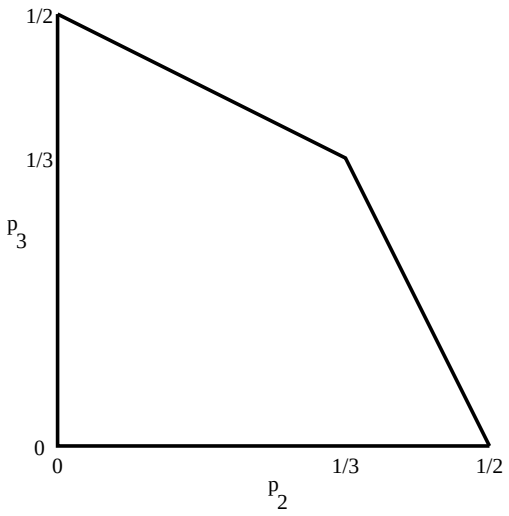
CSC 310, Fall 2011 — Solutions to Theory Assignment #1

Question 1 (15 marks): Consider a source with an alphabet of three symbols, a_1, a_2, a_3, with probabilities p_1, p_2, p_3. Suppose we use a code in which the codewords for a_1, a_2, a_3 are $0, 10, 11$. Describe and plot (by hand is fine) the set of values for p_2 and p_3 for which this is an optimal code. (The value of p_1 is of course determined from these as $p_1 = 1 - p_2 - p_3$.)

Solution: It's easy to see that any optimal binary code for a source with three symbols will have one codeword of length 1 and two codewords of length 2. A uniquely decodeable code with three codewords can't have two of them with length 1, since such a code would not satisfy the McMillian inequality, and making some of the codewords longer would obviously not improve the expected codeword length. So the only question is which of the three symbols should have the codeword of length 1. Clearly, if one symbol has higher probability than the others, it must have this codeword, since otherwise swapping codewords with one of the other symbols would reduce the expected codeword length. If there is a tie for highest-probability codeword, any of them could have the codeword of length 1 in an optimal code.

Accordingly, for the code given in the question to be optimal, it must be the case that $p_1 \geq p_2$ and $p_1 \geq p_3$. This implies that $1 - p_2 - p_3 \geq p_2$ and $1 - p_2 - p_3 \geq p_3$, which implies that $2p_2 + p_3 \leq 1$ and $2p_3 + p_2 \leq 1$. Of course, we also have that $p_2 \geq 0$ and $p_3 \geq 0$. These four linear inequalities define a region that is plotted below:



Question 2 (20 marks): Consider a source with a six-symbol alphabet, a_1, a_2, a_3, a_4, a_5, a_6, with probabilities $p_1 = 0.2$, $p_2 = 0.01$, $p_3 = 0.35$, $p_4 = 0.38$, $p_5 = 0.02$, $p_6 = 0.04$.

a) Find the entropy of this source.

b) Find a Huffman code for this source.

c) Compute the expected code length of this Huffman code.

Solution: (a) The entropy of the source is

$$\begin{aligned} & 0.2\log_2(1/0.2) \;+\; 0.01\log_2(1/0.01) \;+\; 0.35\log_2(1/0.35) \\ & \quad + 0.38\log_2(1/0.38) \;+\; 0.02\log_2(1/0.02) \;+\; 0.04\log_2(1/0.04) \;=\; 1.89 \end{aligned}$$

(b) One possible Huffman code is

a_1	001
a_2	00000
a_3	01
a_4	1
a_5	00001
a_6	0001

(c) The expected codeword length is

$$0.2 \times 3 + 0.01 \times 5 + 0.35 \times 2 + 0.38 \times 1 + 0.02 \times 5 + 0.04 \times 4 \;=\; 1.99$$

Question 3 (20 marks): Suppose we have a source with alphabet $a_1, a_2, \ldots, a_I$, for which the symbols have probabilities $p_1, p_2, \ldots, p_I$. Assume that $p_1 \geq p_2 \geq \cdots \geq p_I$, and that none of the p_i are zero. The coding alphabet is $\{0, 1\}$.

Consider the following conjecture:

> Let C be any uniquely-decodable code for this source in which the code words for $a_1, a_2, \ldots, a_I$ have lengths $l_1, l_2, \ldots, l_I$, and let C' be any other uniquely-decodable code for this source, with codeword lengths of $l'_1, l'_2, \ldots, l'_I$ for $a_1, a_2, \ldots, a_I$. If C and C' are both optimal, then $l_i = l'_i$ for $i = 1, \ldots, I$.

Either prove that this conjecture is true, or demonstrate that it is false by giving an explicit counterexample.

Solution: This question is easier than I intended. An easy counterexample is a source with three symbols, all with probability 1/3. Clearly the code from Question 1 above is optimal, as is any other code with one codeword of length 1 and two codewords of length 2. For example the code C with $a_1 \to 0,\ a_2 \to 10,\ a_3 \to 11$ is optimal, and so is the code C' with $a_1 \to 10,\ a_2 \to 11,\ a_3 \to 0$, but $l_1 = 1$ is not equal to $l'_1 = 2$.

I should have phrased the conjecture as follows:

> Let C be any uniquely-decodable code for this source in which the code words for $a_1, a_2, \ldots, a_I$ have lengths $l_1, l_2, \ldots, l_I$, and let C' be any other uniquely-decodable code for this source, with codeword lengths of $l'_1, l'_2, \ldots, l'_I$ for $a_1, a_2, \ldots, a_I$. If C and C' are both optimal, then there is a permutation $\pi_1, \ldots, \pi_I$ for which $l_i = l'_{\pi_i}$ for $i = 1, \ldots, I$.

In other words, the conjecture says that two optimal codes must have the same set of codeword lengths, but not necessarily the same codeword lengths for any particular symbol.

This conjecture is also false. As a counterexample, consider a source with alphabet a_1, a_2, a_3, a_4 and probabilities $p_1 = 1/3, p_2 = 1/3, p_3 = 1/6, p_4 = 1/6$. The Huffman procedure will merge the two least probable symbols, giving a new symbol with probability 1/3. It will then be arbitrary whether this new symbol is merged with one of a_1 or a_2, or whether instead a_1 and a_2 are merged at this point. The codes that result from either choice will be optimal, but they will not have the same set of codeword lengths (for the first choice, the codeword lengths will be 1, 2, 3, 3 whereas for the second choice the codeword lengths will be 2, 2, 2, 2).

Question 4 (45 marks): Recall the run-length encoding scheme described in the first lecture: We wish to encode a sequence of 500 black or white pixels, in which we expect long runs of black pixels and long runs of white pixels. The first bit we send is 0 if the first run is white, or 1 if the first run is black. We then send a series of 3-bit counts, which are the lengths of runs. If the count is less than 7, the following run is of the opposite colour. For a run with a count of 7 (the maximum specifiable in 3 bits), the next run is the same colour. (Note that a run of exactly 7 pixels would be encoded as a run of length 7 followed by a run of length 0). For example, if the initial pixels are `BBBBBBWWWWBBBBBBBBWWWB...`, the initial bits sent will be 1 110 100 111 001 011...

We can consider this run-length encoding scheme as a way of encoding a single symbol from an alphabet of 2^{500} symbols (one symbol for every possible image with 500 black or white pixels).

The lecture notes give an example of a message with three counts of 3, 0, and 2 pixels (ie, code bits 011 000 010) which will never be produced, since a run of 5 pixels would instead be coded as a single count of 5 (ie, as code bits 101). This illustrates that the code can't be optimal. It can be improved as explained in the lecture notes by getting rid of all nodes in the code tree that have only one child. (There are in fact *two* ways in which this code is inefficient in this respect.)

a) Describe (in as simple terms as you can) what the coding procedure will be once nodes with only one child are eliminated from the code tree.

b) What is the maximum number of bits by which the modification in part (a) shortens a codeword for some image?

c) For this part, assume that the image is not 500 pixels long, but is instead very, very long, so that how a run near the end of the image is coded is of negligible importance. Suppose that the length of a run of white or black pixels is uniformly distributed from 1 to 10 (ie, each of these values has probability 1/10). If the image has n runs (assumed to be very big), what will be the expected number of bits in the encoding if the original code is used, and if the coded as modified in part (a) is used?

d) Find another simple modification to the original code for which the expected number of bits used to encode n runs, using the uniform distribution of run lengths between 1 and 10 as for part (c)), is smaller than for the modified code you found in part (a).

Solution: (a) The code tree for the original procedure has nodes with only one child after two 0 bits following a run of length less than 7 (or for the very first run). This is the situation described in the lecture notes — if a count of less than 7 (ie, 000 to 110) is followed by two 0s, the next bit *must* be a 1, since a run of length zero of the opposite colour would never be produced by the encoder (it would instead have produced a previous count larger than one actually seen). Nodes with only one child also arise near the end of the encoding for an image, when some counts for the length of a run aren't possible because they correspond to an image having more than 500 pixels.

The modified procedure can be described with two tables showing how to encode a run of a certain length in various circumstances. Note that as for the original procedure, runs of length 7 or more are encoded with two or more counts, with the colour of pixel being switched only when a count of length less than 7 is encountered.

Here is the table for use when the previous count was less than 7 (and for the very first run), with the rows corresponding to the count to be encoded, and the columns corresponding to the number of pixels in the image left to code:

count	≥ 7 pixels	6 pixels	5 pixels	4 pixels	3 pixels	2 pixels	1 pixel
1	00	00	00	00	0	0	no bits
2	010	010	010	010	10	1	—
3	011	011	011	011	11	—	—
4	100	100	10	1	—	—	—
5	101	101	11	—	—	—	—
6	110	11	—	—	—	—	—
7	111	—	—	—	—	—	—

And here is the table for use when the previous count was equal to 7, again with the rows corresponding to the count to be encoded, and the columns corresponding to the number of pixels in the image left to code:

count	≥ 7 pixels	6 pixels	5 pixels	4 pixels	3 pixels	2 pixels	1 pixel
0	000	000	000	000	00	00	0
1	001	001	001	001	01	01	1
2	010	010	010	010	10	1	—
3	011	011	011	011	11	—	—
4	100	100	10	1	—	—	—
5	101	101	11	—	—	—	—
6	110	11	—	—	—	—	—
7	111	—	—	—	—	—	—

(b) This modification saves the most bits for an image in which all runs are of length one (so there are 500 runs). One bit is saved when encoding each run of length one (the code is 00 rather than 001), except that for the 3rd to last run and 2nd to last run the saving is two bits (code is 0 rather than 001), and for the last run the saving is three bits (code is no bits at all rather than 001). The total saving for this image is therefore 504 bits.

(c) With the original code, a run of length 1 to 6 is encoded using 3 bits, and a run of length 7 to 10 is encoded using 6 bits, so if run lengths are uniformly distributed between 1 and 10, an image with n runs will have an expected code length of $n \times (3 \times 6/10 + 6 \times 4/10) = 4.2n$ bits. With the code modified as in part (a), a run of length 1 is coded in 2 bits, and longer runs are coded as before, so the expected code length is $n \times (2 \times 1/10 + 3 \times 5/10 + 6 \times 4/10) = 4.1n$ bits.

(d) The main inefficiency with the original code is that a count of pixels in a run is coded in 3 bits using the usual binary notation even when a count of zero is not possible. Instead, we can code the count of pixels in the run *minus one* in 3 bits, so the possible counts are 1 to 8 rather than 0 to 7. We do this only if the previous count was not the maximum, however, since if it was a count of zero is possible.

For instance, we would code a run of length 4 as 011, and a run of length 7 as 110. For longer runs, we would code a run of length 8 as 111 000, a run of length 10 as 111 010, and a run of length 16 as 111 111 001.

If run lengths are uniformly distributed between 1 and 10, the expected code length for n runs with this modification is $n \times (3 \times 7/10 + 6 \times 3/10) = 3.9n$ bits.