Multimedia
Module No: CM3106
Laboratory Worksheet Lab 6 (Week 7): Basic Compression Algorithms

Dr. Kirill Sidorov

Aims and Objectives

After working through this worksheet you should be familiar with:

- Entropy coding methods.
- LZW Compression Algorithm.
- Basic transform coding for images.
- The basic use of MATLAB to investigate the above.

None of the work here is part of the assessed coursework for this module.

Basic Compression in MATLAB

1. **Preliminaries**.

- Download basic_compression_lab.zip from the CM3106 web site.
- Uncompress and install in an appropriate MATLAB accessible directory.

2. **Entropy Coding**.

(a)
- Recall the formula for Shannon's entropy and implement it in MATLAB (one liner!). Alternatively...
- Open and examine `ent.m` (computes Shannon entropy for a string or a vector) and `probs.m` (estimates probabilities by counting frequencies). To get some intuition, try computing entropies for your own strings, chaotic as well fairly regular *e.g.* `AAABAAABCC`...
  What observations can you make?
- Open, examine, and run `coin.m` which plots the entropy of a biased coin as a function of bias, thus convincing yourself that the entropy is highest (1 bit) when the coin is fair.
  Optional: write a piece of code to plot the entropy of a three-sided die (with some probabilities $p_1 + p_2 + p_3 = 1$), in a similar fashion. Since this will be a function of two variables (biases), you may need to use the `surf` command for plotting. When is the entropy the highest? What about n-sided die?

(b) For symbols with frequencies $\{6, 2, 6, 3, 12\}$ *manually* work out codewords using Shannon-Fano and Huffman algorithms. Verify your solution in MATLAB using the provided `huffman.m` and `shannon_fano.m`. Read through these functions and compare their operation with your lecture notes.

(c) Investigate how economically can English text be encoded with these algorithms. Assume, as an approximation, a basic 0-order Markov model for English (*i.e.* all letters are i.i.d.) and for simplicity assume alphabet A...Z with no special characters. The letter frequencies for English are found in `engfreq.m`.

- Build a codeword dictionary using Huffman algorithm, hence compute how many ones and zeros does it take, on average, to encode a letter using Huffman.
- Compare this result with Shannon's limit.
- Build a dictionary using Shannon-Fano algorithm and compare its efficiency with Huffman.
- *Optional:* repeat the same experiment for Klingon.

(d) Investigate how economical Morse code is.

- Compute how many dots/dashes are required on average to encode a letter using Morse code (you will find Morse code lengths in `engfreq.m`).
- Compare your result with the above results for Huffman. What do you observe? Why is this the case?

    - How would you make the comparison more fair (Hint: what is the fundamental difference between Morse code and Huffman code?).

(e) Open `arith_vs_huffman.m` and read through it. It encodes a string (produced by 0-order Markov process, with frequencies defined by `freq` variable) using Huffman and Arithmetic Coding and compares the resulting lengths.

- Explain the observed differences.
- Try running this example with other frequencies and note the results.
- Can you change the frequencies so as to make Huffman fail more miserably relative to Arithmetic Coding?
- *Optional:* take a very long English text and compare the performance of Arithmetic Coding vs Huffman. How does it behave as the text gets longer?

3. **LZW Compression**.

(a) In MATLAB, `cd` to the `lzw` directory[1]. The demo file `lzw_demo.m` reproduces the example discussed in the lecture notes.

- Open the file in MATLAB
  - Examine the `lzw_demo.m` code and familiarise yourself with the functions called and the variables used.

(b) Run `lzw_demo.m` example and note the output.

- Examine the `lzw_compress.m` LZW encoder code. Compare this to the pseudocode given in lecture.
- Examine the `lzw_decompress.m` LZW decoder code. Compare this to the pseudocode given in lecture.

(c) Input and encode/decode your own character sequences. Try and design sequences that build up a set of repetitions to see the code working most effectively. Observe and explain the performance of LZW on a string of the same repeated character, *i.e.* `AAAAAA`. . . .

(d) *Optional:* To better remember the algorithm, pick a string *e.g.* `BANANA_BANDANA` and *manually* compress and decompress it with LZW. Verify your result by running the provided MATLAB code on the same string.

[1] The `lzw_new` directory contains a modified version of this demo with an alternative (easier to read) trace output and uses the initial dictionary of only the characters present in the string.

4. **Basic Transform Coding**.

   (a) Open and examine `simpletrans.m`. This implements the basic example we considered in the lectures: taking advantage of correlation between the colours of adjacent pixels.

   The `simpletransquant.m` demo additionally uses quantisation.

   - Run these demos and explain the output.
   - Plot the histograms (using `hist` command) for the original pixel colours, and for the differences. What do the observed histograms tell you about coding efficiency?
   - Try using different quantisation constants and note the compression ratios vs image quality.

   (b) *Optional:* Examine the following useful transform algorithms (we *did not* cover these in lectures, but it is good to be aware of them. These will not be in the exam.)

   - Burrows-Wheeler transform.
     See http://en.wikipedia.org/wiki/Burrows%E2%80%93Wheeler_transform
     I am providing the implementation in `bwt.m` and (inverse) `ibwt.m`. Investigate whether text pre-transformed with BWT is indeed easier to compress with the algorithms already known to you than the raw text.
   - Try you hand at implementing the move-to-front transform. See http://en.wikipedia.org/wiki/Move-to-front_transform and also answer the question above.