Digital Communication Assignment

Studies on Lempel-Ziv universal data compression (coding)

Student:

Arun Sammit Pandey

Roll no.

17EC30010

Course:

Digital Communication

Aim:

To implement the Universal Lempel-Ziv data compression algorithm (coding) and Huffman compression (coding) algorithm (knowing the TPM) and comparing their performances.

Procedure:

Generation of symbols:

We took a 33X33 Transition probability matrix and used it to generate text file containing symbols from symbol space of cardinality 33. The output was saved in text file named 'symbol_list.txt'

Using the above function, a million (10^6) symbols were generated.

Encoding the symbols using Huffman coding algorithm:

Encoding of symbols was performed using Huffman coding algorithm. For this purpose, we built the Huffman binary tree and found out the optimal codewords from it which was used for encoding the symbols.

Encoding the symbols using Lempel-Ziv algorithm:

We encoded the symbols generated in the first step using LZ algorithm. For doing this firstly we choose the length of dictionary (w). We encoded the first w bits of dictionary trivially. For encoding the remaining symbols, we encoded the reference and length of longest match in dictionary.

Comparison of compression ratios of both methods:

For the purpose of comparing the two methods we found out the compression ratios as:

$$compression\ \% = \frac{bits\ required\ for\ given\ method}{bits\ required\ for\ trivial\ encoding} \times 100$$

Python Code:

Following the previous procedure everything was implemented in python. A brief explanation of different modules/files that were created are as follows:

Symbol_generation.py:

1. checkErgodic(tpm) function:

   Given a tpm it checks whether it is ergodic or not

2. generateSymbols(tpm, total_no_of_symbols, sym_list) function:

This is the main function which generates the symbols given the tpm (transition probability matrix), total_number_of _symbols (total number of symbols) and sym_list (it is the file name in which we want to save the text file of symbols)

Huffman.py:

It is the file in which Huffman coding algorithm is implemented. Its main functions are:

1. _make_initial_heap(p_row):

   This function initializes the heap used for algorithm with a tree with single node containing symbols and probabilities given in p_row.

2. _make_huffman_tree(heap):

   This function merges the two trees with least probabilities on their root node consecutively till only one tree is left inside the heap.

3. _get_huff_codes(root, codes, reverse_codes, current=''):

   This function recursively finds the codewords and reverse-mapping of the codewords from the tree constructed by the function in step 2

4. generate_huffman_codes(all_codes,all_reverse_codes,huffman_forest,tpm):

   This function generates the codes for the whole TPM using the previous function.

5. HuffmanCoding(all_codes, sym_list):

   This function performs the coding of file sym_list and stores the encoded file with name “data_binary_hff.txt”

LZ.py:

1. generateDictionary(w, numbef_of_symbols,sym_list):

   This function generates the dictionary of length w from the file sym_list.

2. generateLZCoding(w, dictionary ,number_of_symbols, sym_list):

   This function performs the LZ coding using the dictionary generated in the previous function and stores the encoded file as “data_binary_Lz.txt”

Observations and Results:

Constant Input Parameters:

Input Parameters	Values
Size of TPM	33
Total number of symbols	1000000

Compression ratio:

Note: w is size of dictionary for LZ coding

Method	% Compression relative to trivial coding
Huffman	59.88 %
LZ (w=50)	50.28%
LZ (w=100)	47.53%
LZ (w=200)	50.41%
LZ (w=500)	62.02%
LZ (w=1000)	73.58%
LZ (w=10000)	89.50%

Discussions:

1. It seems that optimal value of w (size of dictionary) for given value of TPM size and number of symbols is around 100.
2. For this value of w (dictionary size) LZ coding is found to more efficient than Huffman coding.
3. Whereas for w greater than around 500 Huffman coding is more efficient.
4. TPM was generated randomly using a function random.uniform() from numpy package, for different TPM different compression ratios will be observed.
5. If TPM was taken to be of small dimension (less than 4) then LZ coding was seen to perform poorly (compression ratio was close to 100% ie. no significant compression was observed)

Note: To run the codes please put all source codes ie. ‘Huffman.py’, ‘LZ.py’, ‘main.py’ and ‘symbol_list.txt’ in the same folder and then run ‘main.py’ from terminal. After this you will be prompted to enter the size of dictionary (w). Enter its value and wait for some time to see the results.