

Comparative data compression techniques and multi-compression results

M R Hasan¹, M I Ibrahimy², S M A Motakabber³, M M Ferdaus⁴ and M N H Khan⁵

Dept. of Electrical and Computer Engineering, International Islamic University
Malaysia, Gombak, Malaysia

E-mail: rubaiyat.hasan@live.iium.edu.my, ibrahimy@iium.edu.my,
amotakabber@iium.edu.my, ferdaus57@gmail.com and nomanxp76@yahoo.com

Abstract. Data compression is very necessary in business data processing, because of the cost savings that it offers and the large volume of data manipulated in many business applications. It is a method or system for transmitting a digital image (i.e., an array of pixels) from a digital data source to a digital data receiver. More the size of the data be smaller, it provides better transmission speed and saves time. In this communication, we always want to transmit data efficiently and noise freely. This paper will provide some compression techniques for lossless text type data compression and comparative result of multiple and single compression, that will help to find out better compression output and to develop compression algorithms.

1. Introduction

The term “Data Compression” means compresses the data as much as possible from its original size. But for the communication we always want to transmit data very efficiently and noise free. We try to find such data that are lossless in manner. Huffman and LZW both are lossless techniques of data compression. Hybrid compression is also helpful in data compression. There are various hybrid data compression techniques [1, 2].

Text compression algorithms are normally defined in terms of a source alphabet S of 8-bit ASCII codes. Most text compression algorithms perform compression at the character level. If the algorithm is adaptive (as, for example, with any of the Ziv-Lempel methods), the algorithm slowly learns correlations between adjacent pairs of characters, then triples, quadruples and so on. The algorithm rarely has a chance to take advantage of longer range correlations before either the end of input is reached or the tables maintained by the algorithm are filled to capacity. If text compression algorithms were used larger units than single character as the basic storage element, they would be able to take advantage of the longer range correlations and perhaps achieve better compression performance. Faster compression may also be possible by working with larger units.

However, we need to overcome a major problem with text compression algorithms. The number of distinct text words that the compression algorithm has to cope with is, for all practical purposes, unbounded. Thus it makes no sense to implement an algorithm that requires a pre-determined finite alphabet. To use LZW as an example again, we cannot initialize the LZW string table with all sequences of length one, as required in the usual implementations of LZW. Instead, we have to modify the algorithms so that they either pre-determine the set of words used in the source input (an inherently two-pass strategy) or they dynamically expand the source alphabet as each new word occurs. We, of course, advocate single-pass strategies as being more useful for practical applications [3].

The rest of the paper is organized as follows. In Section II, the relevant literature on data compression is reviewed. Section III, shows sample results and discussed about those results. Section IV concludes the paper.

2. Data Compression Techniques

There are two different ways that data compression methods and algorithms can be categorized as the lossless and lossy, the methods are classified according to a fixed or variable. Lossless compressions are run-length; Huffman, delta, LZW etc. and lossy data compression are CS & Q, JPEG and MPEG etc. Here, two lossless compressions Huffman and LZW will be discussed.

- **Huffman Compression Technique**

There are many types of Huffman coding, some are use a Huffman-like algorithm, and others find optimal prefix codes (while, for example, putting different restrictions on the output). In the latter case, the method need not be Huffman-like, and, indeed, need not even be polynomial time. An exhaustive list of papers on Huffman coding and it's variations are given here as follows:

The n-ary Huffman algorithm uses the $\{0, 1, \dots, n - 1\}$ alphabet to encode message and build an n-ary tree. This approach was considered by Huffman in his original paper. The same algorithm applies as for binary (n equals 2) codes, except that the n least probable symbols are taken together, instead of just the 2 least probable. Note that for n greater than 2, not all sets of source words can properly form an n-ary tree for Huffman coding. In this case, additional 0-probability place holders must be added. This is because the tree must form an n to 1 contractor; for binary coding, this is a 2 to 1 contractor, and any sized set can form such a contractor. If the number of source words is congruent to 1 modulo $n-1$, then the set of source words will form a proper Huffman tree.

A variation called adaptive Huffman coding which is involved for calculating the probabilities dynamically based on recent actual frequencies in the sequence of source symbols, and changing the coding tree structure to match the updated probability estimates.

Most often, the weights used in implementations of Huffman coding represent numeric probabilities, but the algorithm which given above does not require this; it requires only the weights form a totally ordered commutative monoid, meaning a way to order weights and to add them. The Huffman template algorithm enables one to use any kind of weights (costs, frequencies, pairs of weights, non-numerical weights) and one of many combining methods (not just addition). Such algorithms can solve other minimization problems, a problem first applied to the circuit design.

Length-limited Huffman coding is a variant where the goal is still to achieve a minimum weighted path length, but there is an additional restriction that the length of each code word must be less than a given constant. The package-merge algorithm solves this problem with a simple greedy approach which is very similar to that is used by Huffman's algorithm. It's time complexity is $O(nL)$, where L is the maximum length of a code word. No algorithm is known to solve this problem in linear or linearithmic time, unlike the pre-sorted and unsorted conventional Huffman problems, respectively [4, 5]

Huffman Encoding Algorithm

Huffman (W, n) //Here, W means weight and n is the no. of inputs

Input: A list W of n (Positive) Weights.

Output: An Extended Binary Tree T with Weights Taken from W that gives the minimum weighted path length.

Procedure: Create list F from singleton trees formed from elements of W.

While (F has more than 1 element) do

Find T1, T2 in F that have minimum values associated with their roots // T1 and T2 are sub tree

Construct new tree T by creating a new node and setting T1 and T2 as its children

Let, the sum of the values associated with the roots of T1 and T2 be associated with the root of T Add T to F

Do

Huffman-Tree stored in F

- **LZW Compression Technique**

LZW compression named after its developers, A. Lempel and J. Ziv, with later modifications by Terry A. Welch. It is the foremost technique for general purpose data compression due to its simplicity and versatility. Typically, you can expect LZW to compress text, executable code, and similar data files to about one-half their original size. LZW also performs well when presented with an extremely redundant data files, such as tabulated numbers, computer LZW is the basis of several personal computer utilities that claim to "double the capacity of your hard drive." If the codeword length is not sufficiently large, Lempel-Ziv codes may also rise slowly to reasonable efficiency, maintain good performance briefly, and fail to make any gains once source code, and acquired signals. Compression ratios of 5:1 are common for these cases. [2]

LZW Encoding Algorithm

Step 1: At the start, the dictionary contains all possible roots, and P is empty

Step 2: C: = next character in the char stream;

Step 3: Is the string P+C present in the dictionary?

(a) if it is, P := P+C (extend P with C);

(b) if not,

–output the code word which denotes P to the code stream;

– add the string P+C to the dictionary;

–P: = C (P now contains only the character C); (c) Are there more characters in the char stream?

–if yes, go back to step 2;

–if not:

Step 4: Output the code word which denotes P to the code stream;

Step 5: END.

The fundamental theory of LZW compression algorithm is: any predictable data can demonstrate such predictability by certain mark and shorten the data length. During the process of LZW compression, as the length of each code in the code stream of a datum produced after compression is less than N, or the compression algorithm can represent number of $0 \sim 2^n - 1$, therefore the string list can accommodate number of 2^n at the maximum. But each element of the input data stream after compression is a byte, which represents $0 \sim 255$ possible assignments. Secondly, compression program takes one byte from the input character stream. Two buffers of current prefix code and current string are used to store data. Prefix position is for the code processed last time, current string is for the character string represented by prefix code and characters read just now. When the program starts, both the prefix code and current string are blank. Thirdly, program searches for current string in the string list after initialization.

Fourthly, read the next byte from input stream, and add this character behind current string, now the current string contains 2 bytes, and repeat the third step.

Obviously double-byte characters are unable to find in the string list this time, because all the existing characters are all single characters in the string list. So the program turns to another branch, and establishes a new item in the string list responding to the current string of double byte characters, or the 256th item, and then adds the prefix code to output stream, therefore the compressed codes we need would be available. As the maximum number the string list can accommodate is $2n$, when the string list capacity is nearly full, enable the cleaning code and restart initialization for prefix code and current string, and repeat from step one to step four to compress data [6].

Table 1. Comparative Data Compression Table.

Sample Name	Original Data	LZW Comp.	Compression ratio	Huffman Compression	Comp. ratio	LZW based Huffman	Comp. ratio	Huffman based LZW	Comp. ratio
test1	50.39 MB	10.22MB	4.42	30.01 MB	1.79	20.24 MB	2.41	10.20 MB	4.49
test2	43.1 MB	9.75 MB	4.42	24.1 MB	1.79	9.72 MB	4.43	9.65 MB	4.46
test3	12.7 MB	3.22 MB	3.94	7.20 MB	1.76	5.6 MB	2.27	3.10 MB	4.09
test4	81.4 MB	19.4 MB	4.19	45.7 MB	1.78	28.2 MB	2.89	19.4 MB	4.19
test5	20.82 MB	10.80 MB	3.52	10.76 MB	1.6	10.27 MB	2.22	10.74 MB	3.81
test6	110.3 MB	30.18 MB	3.55	70.07 MB	1.59	50.78 MB	1.95	20.35 MB	4.81
test7	86.3 MB	19.4 MB	4.45	48.3 MB	1.79	35.8 MB	2.41	19.2 MB	4.49
test8	32 MB	7.5 MB	4.27	15.8 MB	2.02	11.5 MB	2.78	7.1 MB	4.51
test9	14.7 MB	3.1 MB	4.74	7.1 MB	2.07	6.8 MB	2.16	3.00 MB	4.9
test10	126.5 MB	30.1 MB	4.2	58.5 MB	2.16	53.7 MB	2.35	28.7 MB	4.41
Average			4.17		1.835		2.587		4.416

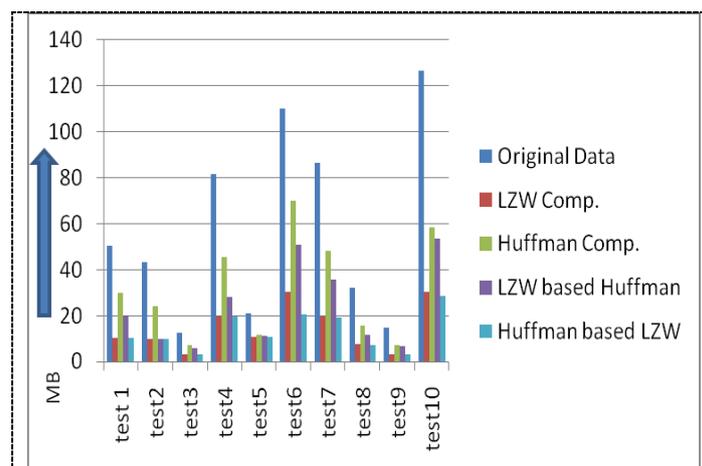


Figure 1. Comparative Compression ratio from the data.

Proposed Algorithm

Step 1: At the start, the dictionary contains all possible roots, and P is empty

Step 2: C: = next character in the char stream;
 Step 3: Is the string P+C present in the dictionary?
 (a) if it is, P := P+C (extend P with C);
 (b) if not,
 –output the code word which denotes P to the code stream;
 – add the string P+C to the dictionary;
 –P: = C (P now contains only the character C); (c) Are there

more characters in the char stream?

–if yes, go back to step 2;

–if not:

Step 4: Output the code word which denotes P to the code stream;

Step 5: Take, W means weight and n is the no. of inputs, A

list W of n (Positive) Weights, An Extended Binary Tree T with Weights Taken from W that gives the minimum Weighted path length.

Procedure: Create list F from singleton trees formed from elements of W.

While (F has more than 1 element) do

Find T1, T2 in F that have minimum values associated with their roots. // T1 and T2 are sub tree

Construct new tree T by creating a new node and setting T1 and T2 as its children

Let, the sum of the values associated with the roots of T1 and T2 be associated with the root of T Add T to F

Do

Step 6: END.

3. Results and discussion

Here, the compression ratio (Cr) measured as, $Cr = \text{Original data size} / \text{Compressed data size}$

- Data Compression using Huffman: From the table 1 and figure 1 in case of Huffman experiment this is found that every data has decreased from its original size. For the all 10 cases of compressions the compression ratio is about 1.84 in average case. There are lower and higher compression ratios in some cases. This is the result of compression of text file. The compressed files were in the 0, 1 binary code formats. After that when decompressed that file it becomes same. No data loss is found there.
- Data Compression using LZW: From table 1 and figure 1 in case of the LZW experiment it is found that the data decreased from its original size. The compression ratio is 4.17 in average case. There are lower and higher compression ratios in some cases. This is the result of compression of text file. When decompressed that file it becomes same as the original file. It gives better compression ratio than the case of Huffman Encoding. It compresses a file almost 78% and we can have only 22% size of the original file size. To have better file size we may use LZW Encoding technique.
- Data Compression ratio using LZW based Huffman: Here, in table 1 and figure 1 it is found that every data has decreased from its original size. The compression ratio is 2.59 in average case. There are several lower and higher compression ratios in some cases. After that when decompressed that file it becomes same as the original file. It gives better compression ratio than the case of Huffman Encoding but lower compression ratio in the case of LZW. It compresses a file almost 66% and we can have only 34% size of the original file size. We think that Hybrid compression gives better output but here we find different case. So, LZW performs better than LZW based Huffman Encoding. To have better file size we may use LZW Encoding technique.

- Data Compression using Huffman based LZW: From the table 1 and figure 1 of first Huffman and then LZW experiment it is clear that every data has decreased from its original size. From the 10 sample the compression ratio 4.42 in average case. There are lower and higher compression ratios in some cases. This is the result of compression of text file. After that when we have decompressed that file it becomes same as the original file. It gives better compression ratio than the case of Huffman, LZW and LZW based Huffman all three Encoding. It compresses a file almost 79% and we can have only 21% size of the original file size. From this experiment we can say that Hybrid Data Compression is better than single compression. To have better file size we may use Huffman based LZW Encoding technique.
- Compare with Recent Approaches: Recently, WinRar is widely used for data compression. The algorithm is based on Lempel-Ziv (LZSS) and prediction by partial matching (PPM) compression is used for WinRar. The average compression ratio achieved for the set of test images was 70%, the image size was usually reduced to 30% of their original value [7]. Here, we introduced multi-compression algorithm for Text data only. Significant result is that it compresses data at about 78-79% form Table 1. So, if work with this algorithm and improve for all sort of data then there are some prospect to get better compression ratio.

4. Acknowledgement

This paper is the enlarged and modified concept of Data Compression using Huffman based LZW Encoding Technique (IJSER, Vol- 2, Issue 11, Nov-2011), published at International Journal of Scientific & Engineering Research conference. In that paper there was limitations in compression ratio, here the better compression ratio is found.

5. Conclusion

Data compression is a topic of much importance and many applications. Methods of data compression have been studied for almost four decades. This paper has provided multi-data compression methods of general utility. The proposed algorithm has been evaluated in terms of the amount of compression ratio they provided, algorithm efficiency, and susceptibility to error. While algorithm efficiency and susceptibility to error are relatively independent of the characteristics of the source ensemble, the amount of compression achieved depends upon the characteristics of the source to a great extent. It does not matter how the characters are arranged. It arranged above so that the final code tree looks nice. It is used Turbo C editor for coding. It does not matter how the final code tree are labelled (with 0s and 1s).It has chosen to label the upper branches with 0s and the lower branches with 1s. In conclusion anybody can say that Huffman based LZW Encoding can Compresses data more than all other three cases, when in average case the Huffman based LZW compression ratio is 4.41, where other max average compression ratio is 4.17 in case of LZW compression. The Huffman based LZW compression is better some of the cases than LZW Compression. This is also possible to Decompress.

6. References

- [1] Rafael C. Gonzalez and Richard E. Woods, Digital Image Processing (Second papers: Edition)-2006, Pages: 411, 440-442, 459
- [2] M Rubaiyat Hasan, Data Compression using Huffman based LZW Encoding Technique, International Journal of Scientific & Engineering Research, Vol- 2, Issue 11, Nov-2011.
- [3] R. Nigel Horspool, Gordon V. Cormack, Constructing Word-Based Text Compression Algorithms, IEEE 1992.
- [4] Suresh Kumar, Madhu Rawat, Vishal Gupta, Satendra Kumar, The Novel Lossless Text Compression Technique Using Ambigram Logic and Huffman Coding, Information and Knowledge Management ISSN 2224-5758 (Paper) , Vol -2, No.2, 2012.

- [5] Julia Abrahams, Code and Parse Trees for Lossless; Source Encoding, IEEE 1998
- [6] YuanJing, The Combinational Application of LZSS and LZW Algorithms for Compression Based On Huffman, International Conference on Electronics and Optoelectronics (ICEOE 2011), 2011.
- [7] Tatjana LonCar-Turukalo, Vladimir CrnojeviC, ieljen Trpovski, Image Compression by Decomposition into Bit Planes, IEEE- 2001