New Data Compression Algorithm and its Comparative Study with Existing Techniques

Rakesh
Waghulde
Information
Technology Dept
V.J.T.I., Mumbai –
400019

Harshal Gurjar
Information
Technology Dept.
V.J.T.I., Mumbai –
400019

Vishal Dholakia
Information
Technology Dept
V.J.T.I., Mumbai –
400019

G.P. Bhole
Head of department
Information
Technology Dept
V.J.T.I., Mumbai –
400019

ABSTRACT

Data compression is a technique to represent data using less number of bits than original data. There are various data compression techniques available, but still there is a need to achieve more compression ratio. This paper proposes an algorithm that combines the features of both Huffman's algorithm and LZW algorithm to achieve more compression ratio. This algorithm is named as VJ Zip. In the new algorithm VJ Zip, for compression, firstly every duplicate occurrence of data is replaced with the pointer to its previous occurrence to obtain partially compressed data. From this partially compressed data, the literals and pointers are further compressed using two separate Huffman trees. We measure the performance of this new algorithm in terms of compression ratio and also compare the performance of this new modified algorithm with the two algorithms viz., Huffman's algorithm and LZW algorithm, individually. Comparing the results it is inferred that new modified algorithm, VJ Zip, is more efficient than Huffman's algorithm and LZW algorithm applied individually. On an Average, it achieves 26% & 54% more compression ratio for .txt and .xml format respectively, as compared to Huffman's algorithm and 16% & 18% more compression ratio for .txt and .xml format respectively, as compared to LZW. Also this paper compares the performance of new algorithm with the existing software 7Zip. As compared to 7Zip new modified algorithm gives almost same compression ratios for text format while achieves 1% more compression ratio for images and videos.

Keywords

Data compression, decompression, compression ratio, efficiency, encoding, decoding. (Keywords)

1. INTRODUCTION

Data compression is the technique used to reduce the redundancies in data representation in order to decrease data storage requirements and hence communication costs.[1] It also helps to fit more files into a limited amount of space. Reducing the storage requirement also increases the capacity of storage medium and communication bandwidth.. Compression is possible because most real-world data is redundant and contains repetition in data. Compression is the need of the day, may it be multimedia applications, transmission applications or for that other application all need data to be stored in a compact way for minimum cost of storage and transmission. There are large numbers of compression techniques available which are capable of application specification to large compression ratios [1]. Here we limit our scope to the LZ77 algorithm, LZW algorithm and Huffman's algorithm which can be considered as the industry

standard for the lossless data compression. Data compression can be explained as a process that takes an input data or original data (D) and represents it in compact form comp (D) using less number of bits as compared to input data thus reducing the storage size of input data. Reverse of this process is known as data decompression where the compressed data comp (D) is used to regenerate the original data or input data (D) as shown in Fig 1. Sometimes this system of compression (coding) and decompression (decoding) is together referred to as a CODEC.[1]

Fig 1:- CODEC

Based on similarity between decompressed data and original data, data compression can be classified into two types viz. lossless data compression and lossy data compression. After decompression, if the decompressed data and original data are exactly same then the process is known as lossless data compression. On the other hand if decompressed data and original data are not exactly same then the process is known as lossy data compression. Lossless data compression techniques are generally applied on critical data such as text data, XML data or scientific data. Sometimes the terms lossless data compression and lossy data compression are also referred to as noiseless data compression and noisy data compression respectively, where the term noise stands for the error in the reconstruction of data because the reconstructed data is not exactly same as the original one. Data compression methods can also be categorized as static coding or dynamic coding. For static coding, particular set of input data is always mapped to same compressed code while in case of dynamic method mapping between input data to compressed code may change over time. Dynamic coding is also known as adaptive data compression technique, since, compressed code adapts to the changes in the input data. Usually frequency or probability of occurrence of symbol is taken in account for constructing compressed code. An adaptive formulation of the binary code words of the symbols is suitable when the probabilities or frequency of occurrences of the symbols from the source are not fixed over time [2].

2. DIFFERENT COMPRESSION TECHNIQUES

There are various compression techniques available. We will get an overview of some basic and commonly used techniques.

2.1 LZ77 Algorithm

The LZ77 Compression Algorithm is used to analyze input data and determine how to reduce the size of that input data by replacing redundant information with metadata. Every duplicate occurrence of data is replaced with the pointer to its first occurrence. Pointer here is a metadata that contains the information about how to expand or decompress that section again. LZ77 uses sliding window to compare the data [3].

2.2 LZW

LZW is a dictionary based algorithm used for data compression. Firstly, dictionary is initialize to contain the all the strings with length one. Longest match for the current input is searched in the dictionary. If the match is found, then it is replaced with the index of dictionary. also the match is added to dictionary with new dictionary index. this process is continued till the end of input string. GIF file format uses LZW algorithm.

2.3 7-Zip

7zip is the new archive format, providing high compression ratio [5]. The main features of 7zipformat:

- Open architecture
- High compression ratio
- Strong AES-256 encryption
- Supporting files with sizes up to 16000000000 GB
- Efficient Compression Ratio

2.4 Huffman Algorithm

Huffman coding is a lossless technique for compressing input data. The concept of Huffman's algorithm is to assign variable length codes to the input characters. The frequency of occurrence of that character in input stream decides the length of the character. The character with highest frequency of occurrence is mapped to the smallest code, while character with lowest frequency of occurrence is mapped to the largest code [6]. Also codes are constructed in such a way that no code assigned to a character is prefix to any other character. This ensures the unambiguous decoding of the generated code sequence [7].

3. PROPOSED ALGORITHM(VJ-ZIP)

Proposed algorithm (VJ-Zip) uses combined advantages of LZW algorithm, Huffman's algorithm and hash table.

Firstly, VJ-Zip finds the duplicate strings in the input data. (Here, string stands for arbitrary sequence of byte and not the printable characters). Every second occurrence of string is replaced with the pointer to its previous occurrence. The format of (distance, length) is used to represent this pointer, where distance is a distance to be travel to left of the string to reach the starting position of first occurrence of string and length represents the length of string matched. For VJ-Zip distance is limited to 32bytes while length of string is limited to 256 bytes. If string is not found in previous 32 bytes then it is considered as a new string or literal bytes. Now literals and match distances are compressed using two separate Huffman trees. At the start of each block these trees are stored in a compact form. There is no restriction on size of block but the compressed data for one

block must fit in available memory. A block is terminated when it's useful to start another block with fresh trees.

Hash table is used to find duplicate strings. Entire input string is divided into block of length 3. All such block, one at a time is inserted in hash table. Hash index for each block is computed. Now if hash chain for any index in non-empty, then the input string is compared with the entire string in that chain and longest match of string is selected. This search begins with the most recent string, so as to favor small distances.

Fig 2:-Block diagram

Singly link list data structure is used to represent the hash chain. There are no deletions from the hash chains; the algorithm simply discards matches that are too old. To avoid a worst-case situation, very long hash chains are arbitrarily truncated at a certain length, determined by a runtime option. So the algorithm does not always find the longest possible match but generally finds a match which is long enough. After a match of length L has been found, algorithm searches for a longer match at the next input byte. If a longer match is found, the previous match is truncated to a length of one (thus producing a single literal byte) and the process of comparison begins again. Otherwise, the original match is kept, and the next match search is attempted only N steps later. New strings are inserted in the hash table only when no match was found, or when the match is not too long. This degrades the compression ratio but such reduction in compression ratio is very negligible but on the other hand it saves considerable time, thus making the comparison faster.[1]

Fig 3:- Compressed Data

4. COMPARATIVE STUDY

The table given below shows the comparison of the compression algorithms on Text file (.txt file)

Table1:- Comparison of compression algorithms on .txt files

File	Actual Size(KB)	VJ-Zip		LZW		7-ZIP		Huffman	
		Compressed Size(KB)	%Ratio	Compressed Size(KB)	%Ratio	Compressed Size(KB)	%Ratio	Compressed Size(KB)	%Ratio
T1	45	13	71.11	20	55.55	13	71.11	26	42.22
T2	156	46	70.51	66	57.69	42	73.07	83	46.79
T3	294	102	65.30	146	50.34	93	68.36	166	43.53
T4	407	151	62.89	202	50.36	137	66.33	240	41.03
T5	521	175	66.41	285	45.29	155	70.24	292	43.95
T6	584	206	64.72	283	51.54	182	68.83	339	41.95
T7	688	238	65.40	334	51.45	205	70.20	390	43.31
T8	855	308	63.97	417	51.22	267	68.77	486	43.15
T9	1037	367	64.60	522	49.66	307	70.39	600	42.14
T10	1149	419	63.53	689	40.03	354	69.19	655	42.99
T11	1178	368	68.76	600	49.06	301	74.44	671	43.03
T12	3143	1115	64.52	1564	50.23	902	71.30	1784	43.23
T13	4315	1235	71.37	2143	50.33	995	76.94	2490	42.29
T14	6463	2418	62.58	3261	49.54	1919	70.30	3690	42.90
T15	12482	4470	64.18	6511	47.83	3512	71.86	7159	42.64

Fig 4:- Text file compression comparison by various algorithms

Conclusion from the above graph: From the statistics, it can be inferred that for text file compression, VJ-zip algorithm is more efficient than LZW and Huffman algorithm applied individually.

The table given below shows the comparison of the compression algorithms on XML file (.xml file)

Table2 :- Comparison of Compression algorithms on XML files

File	Actual Size(KB)	VJ Zip		LZW		7-ZIP		Huffman	
		Compressed Size(KB)	%Ratio	Compressed Size(KB)	%Ratio	Compressed Size(KB)	%Ratio	Compressed Size(KB)	%Ratio
X1	2	1	50	1	50	1	50	2	0
X2	4	1	75	2	50	1	75	3	25
X3	16	2	87.5	5	68.75	2	87.5	11	31.25
X4	17	2	88.23	6	64.70	2	88.23	11	35.29
X5	17	2	88.23	6	64.70	2	88.23	11	35.29
X6	147	4	97.27	36	75.51	4	97.27	92	37.41
X7	239	20	91.63	51	78.64	18	92.46	144	39.74
X8	2188	100	95.42	582	73.40	100	95.42	1429	34.68

Fig 5:- XML file comparison by various algorithms.

Conclusion from the above graph: From the statistic, it can be inferred that for XML file compression, VJ-zip algorithm is more efficient than LZW and Huffman algorithm applied individually.

The table gives below shows the comparison of the compression algorithms on some other file formats (.jpg, .mp3 file)

Table3:- Comparison of compression algorithms on .jpg and .mp3 files

File	Actual Size(KB)	VJ-Zip		7-ZIP		Huffman	
		Compressed Size(KB)	%Ratio	Compressed Size(KB)	%Ratio	Compressed Size(KB)	%Ratio
ij1	1561	1556	0.32	1567	-3.84	1562	-0.064
ij2	3583	3575	0.22	3606	-0.60	3585	-0.055
lj3	3773	3766	0.18	3799	-0.64	3777	-0.1
1mp3	4730	4418	5.33	4474	5.41	4655	1.58
2mp3	8218	8063	1.89	8064	1.87	8204	0.17

Fig 6 :- Image and mp3 compression by various algorithm.

From the statistics, it can be inferred that VJ-zip algorithm is more efficient for the image and mp3 files than the Huffman, 7-zip algorithm.

4.1 Limitations of existing system from above statistics

- Compression ratio of the Huffman & LZW algorithm is less than VJ-zip algorithm.
- In case of compression of image and audio files, 7-zip is giving negative result in some cases while proposed algorithm gives efficient result.

4.2 Advantages of proposed algorithm

- Proposed algorithm is efficient for files with repeated data
- For xml files , considerable compression ratios up to 97% can be achieved
- It combines the advantages of Huffman algorithm , hash table mapping and LZW

4.3 Limitations of proposed algorithm

- Propose Algorithm is not efficient for small file up to 2KB.
- Propose algorithm is inefficient when no data in file is repeated.

5. CONCLUSION

We have implemented and studied all the above algorithms and their concepts. From these studies, we have concluded that:

- Huffman algorithm works on all the file formats which uses lossless compression.
- LZ77 and LZW are more effective on text file formats
- Huffman algorithm and lz77 algorithm are more efficient for file formats which can undergo lossless compression (.txt, .xml, .docx) than those with lossy compression (.jpeg, .gif, .mp3, .mp4).
- Proposed algorithm, VJ Zip, combines the feature of Huffman algorithm and lzw.

- As compared to Huffman's algorithm, proposed algorithm achieves 26% & 54% more compression ratio for .txt and .xml format respectively
- As compared to LZW algorithm, proposed algorithm achieves 16%& 18% more compression ratio for .txt and .xml format respectively.
- Results of proposed algorithm are best achieved on XML file format(.xml).An average of 91% compression ratio can be achieved for XML file format
- For proposed algorithm, Repetition of data in file increases the compression ratio.
- Compression Techniques are more suitable for large data sizes.

6. REFERENCES

- [1] Khalid Sayood, "Introduction to Data Compression", 3rd edition.
- [2] ShrutiPorwal, YashiChaudhary, Jitendra Joshi, Manish Jain, "Data Compression Methodologies for Lossless Data and Comparison between Algorithms", International Journal of Engineering Science and Innovative Technology (IJESIT), ISSN: 2319-5967, Volume 2, Issue 2, March 2013
- [3] "LZ77 and LZ78" Wikipedia, [online]Available: http://en.wikipedia.org/wiki/LZ77_and_LZ78
- [4] "Lempel–Ziv–Welch" Wikipedia,[online]Available: <http://en.wikipedia.org/wiki/Lempel%E2%80%93Ziv%E2%80%93Welch>
- [5] "7z", Wikipedia,[online]Available: <http://en.wikipedia.org/wiki/7z>
- [6] "Huffman coding",[online]Available: http://en.wikipedia.org/wiki/Huffman_coding
- [7] "Greedy algorithm (Huffman coding)-set3",[online]Available: <http://www.geeksforgeeks.org/greedy-algorithms-set-3-huffman-coding/>