Analysis of Lossless Data Compression Techniques

Dalvir Kaur[1] , Kamaljeet Kaur[2]
[1]Master of Technology in Computer Science & Engineering, Sri Guru Granth Sahib World University, Fatehgarh Sahib, Punjab, India.
[2]Assistant Professor, Department Of Computer Science & Engineering, Sri Guru Granth Sahib World University, Fatehgarh Sahib, Punjab, India.

Abstract:
Compression is useful to reduce the size of data .There are different compression algorithms which are available in different formats.Data compressions are generally lossless and lossy data compression. In this paper, we study different methods of lossless data compression algorithms like- Shanon-Fano coding, Huffman Encoding, Run-Length Encoding (RLE), Lempel-Ziv-Welch (LZW).

Keywords: *Data compression, Huffman Coding, Lempel-Ziv, Lossless Compression, Lossy Compression, Shanon-fano coding.*

I. INTRODUCTION

Compression is the reduction in size of data by converting it to a format that requires fewer bits. Most often compression is used to minimize storage space (on a hard drive, for example) or for reducing transmitted data over a network. In other words, Compression is the art of representing the information in a compact form rather than its original run compressed form. In other words, using the data compression, the size of a particular file can be reduced.[2] Data compression refers to reducing the amount of space needed to store data or reducing the amount of time needed to transmit data. The size of data is reduced by removing the excessive Information. Data compression can be *lossless*, only if it is possible to exactly reconstruct the original data from the compressed version [4]. To compress something means that you have a piece of data and you decrease its size.
There are different data compression techniques who to do that and they all have their own advantages and disadvantages. Examples of such source data are medical images, text and images Preserved for legal reason, some computer executable files, etc. The general principle of data compression algorithms on text files is to transform string of characters into a new string which contains the same information but with new length as small as possible. The efficient data compression algorithm is chosen according to some scales like: compression size, compression ratio, processing time or speed, and entropy. [1]

1.1. Lossless compression vs lossycompression:

Lossless compression: Reduces bits by identifying and eliminating statistical redundancy. no information is lost in lossless compression. In these schemes before the compression after the compression data must be same.

1.2 Lossy compression:

Reduces bits by identifying marginally important information and removing it. In these schemes some loss of information is acceptable depending upon the application [2].

Compression ratio=B1/B0*100%.

B0=no. of bits before compression.
B1= no. of bits after compression

II. Lossless data Compression Algorithm:

Lossless data compression is the size reduction of a file, such that a decompression function can restore the original file exactly with no loss of data. Lossless data compression is used ubiquitously in computing, from saving space on your personal computer to sending data over the web, communicating over a secure shell, or viewing a PNG or GIF image. The basic principle that lossless compression algorithms work on is that any non-random file will contain duplicated information that can be condensed using statistical modeling techniques that determine the probability of a character or phrase appearing. These statistical models can then be used to generate codes for specific characters or phrases based on their probability of occurring, and assigning the shortest codes to the most common data. Such techniques include Huffman coding , run-length encoding

Lempel-ziv Algorithm,Shanon-fano coding and, . Using these techniques and others, an 8-bit character or a string of such characters could be represented with just a few bits resulting in a large amount of redundant data being removed.[8]

III. SHANON-FANO CODING

In Shannon–Fano coding, the procedure is done by a more frequently occurring string which is encoded by a shorter encoding vector and a less frequently occurring string is encoded by longer encoding vector. Shannon-Fano coding. Relied on the occurrence of each character or symbol with their frequencies in a list and is also called as a variable length coding. The Shannon-Fano Algorithm This is a basic information theoretic algorithm [3].

IV. RUN-LENGTH ENCODING (RLE)

Run Length Encoding (RLE) is a simpleand popular data compression algorithm. It is based on the idea to replace a long sequence of the same symbol by a shorter sequence and is a good introduction into the data compression field for newcomers.RLE requires only a small amount of hardware and software resources. Therefore RLE was introduced very early and a large range of derivates have been developed up to now. Run-length encoding is a data compression algorithm that is supported by most bitmap file formats, such as TIFF, BMP, and PCX[2].

V. Huffman Coding

Huffman Encoding Algorithms use the probability distribution of the alphabet of the source to develop the code words for symbols. The frequency distribution of all the characters of the source is calculated in order to calculate the probability distribution. According to the probabilities, the code words are assigned. Shorter code words for higher probabilities and longer code words for smaller probabilities are assigned. For this task a binary tree is created using the symbols as leaves according to their probabilities and paths of those are taken as the code words. Two families of Huffman Encoding have been proposed: Static Huffman Algorithms and Adaptive Huffman Algorithms. Static Huffman Algorithms calculate the frequencies first and then generate common tree for both the compression and decompression processes . Details of this tree should be saved or transferred with the compressed file. The Adaptive Huffman algorithms develop the tree while calculating the frequencies and there will be two trees in both the processes. In this approach, a tree is generated with the flag symbol in the beginning and is updated as the next symbol is read[1].

VI. LEMPEL-ZIV ALGORITHM:

Data compression up until the late 1970's mainly directed towards creating better methodologies for Huffman coding. An innovative, radically different method was introduced in1977 by Abraham Lempel and Jacob Ziv. This technique (called Lempel-Ziv) actually consists of two considerably different algorithms, LZ77 and LZ78[7].

Due to patents, LZ77 and LZ78 led to many variants:

LZ77 Variants	LZR	LZSS	LZB	LZH		
LZ78 Variants	LZW	LZC	LZT	LZMW	LZJ	LZFG

6.1 LZ77 Sliding Window Algorithms

Published in 1977, LZ77 is the algorithm that started it all. It introduced the concept of a 'sliding window' for the first time which brought about significant improvements in compression ratio over more primitive algorithms. LZ77 maintains a dictionary using triples representing offset, run length, and a deviating character. The offset is how far from the start of the file a given phrase starts at, and the run length is how many characters past the offset are part of the phrase. The deviating character is just an indication that a new phrase was found, and that phrase is equal to the phrase from offset to offset+length plus the deviating character. The dictionary used changes dynamically based on the sliding window as the file is parsed. For example, the sliding window could be 64MB which means that the dictionary will contain entries for the past 64MB of the input data[7].Given an input "abbadabba" the output would look something like "abb(0,1,'d')(0,3,'a')" as in the example below:

Symbol	Output
a	a
b	b
b	b
a	(0, 1, 'd')
d	
a	(0, 3, 'a')
b	
b	
a	

While this substitution is slightly larger than the input, it generally achieves a considerably smaller result given longer input data.

6.2 LZR

LZR is a modification of LZ77 invented by Michael Rodeh in 1981. The algorithm aims to be a linear time alternative to LZ77. However, the encoded pointers can point to any offset in the file which means LZR consumes a considerable amount of memory. Combined with its poor compression ratio (LZ77 is often superior) it is an unfeasible variant[7].

6.3 LZSS

The LZSS, or Lempel-Ziv-Storer-Szymanski algorithm was first published in 1982 by James Storer and Thomas Szymanski. LZSS improves on LZ77 in that it can detect whether a substitution will decrease the filesize or not. If no size reduction will be achieved, the input is left as a literal in the output. Otherwise, the section of the input is replaced with an (offset, length) pair where the offset is how many bytes from the start of the input and the length is how many characters to read from that position .Another improvement over LZ77 comes from the elimination of the "next character" and uses just an offset-length pair[7]. Here is a brief example given the input " these theses" which yields " these(0,6)s" which saves just one byte, but saves considerably more on larger inputs.

Index	0	1	2	3	4	5	6	7	8	9	10	11	12
Symbol		t	h	e	s	e		t	h	e	s	e	s
Substituted		t	h	e	s	e	(	0	,	6	)	s	

LZSS is still used in many popular archive formats, the best known of which is RAR. It is also sometimes used for network data compression.

6.4 LZH

LZH was developed in 1987 and it stands for "Lempel-Ziv Huffman." It is a variant of LZSS that utilizes Huffman coding to compress the pointers, resulting in slightly better compression. However, the improvements gained using Huffman coding are negligible and the compression is not worth the performance hit of using Huffman codes[7].

6.5 LZB

LZB was also developed in 1987 by Timothy Bell et al as a variant of LZSS. Like LZH, LZB also aims to reduce the compressed file size by encoding the LZSS pointers more efficiently. The way it does this is by gradually increasing the size of the pointers as the sliding window grows larger. It can achieve higher compression than LZSS and LZH, but it is still rather slow compared to LZSS due to the extra encoding step for the pointers[7].

6.6 LZ78 Dictionary Algorithms:

LZ78 was created by Lempel and Ziv in 1978, hence the abbreviation. Rather than using a sliding window to generate the dictionary, the input data is either preprocessed to generate a dictionary wiith infinite scope of the input, or the dictionary is formed as the file is parsed. LZ78 employs the latter tactic. The dictionary size is usually limited to a few megabytes, or all codes up to a certain numbers of bytes such as 8; this is done to reduce memory requirements. How the algorithm handles the dictionary being full is what sets most LZ78 type algorithms apart. While parsing the file, the LZ78 algorithm adds each newly encountered character or string of characters to the dictionary. For each synbol in the input, a dictionary entry in the form (dictionary index,

unknown symbol) is generated; if a symbol is already in the dictionary then the dictionary will be searched for substrings of the current symbol and the symbols following it. The index of the longest substring match is used for the dictionary index. The data pointed to by the dictionary index is added to the last character of the unknown substring. If the current symbol is unknown, then the dictionary index is set to 0 to indicate that it is a single character entry. The entries form a linked-list type data structure.

An input such as "abbadabbaabaad" would generate the output "(0,a)(0,b)(2,a)(0,d)(1,b)(3,a)(6,d)". You can see how this was derived in the following example:

Input:		a	b	ba	d	ab	baa	baad
Dictionary Index	0	1	2	3	4	5	6	7
Output	NULL	(0,a)	(0,b)	(2,a)	(0,d)	(1,b)	(3,a)	(6,d)

6.7 LZW

LZW is the Lempel-Ziv-Welch algorithm created in 1984 by Terry Welch. It is the most commonly used derivative of the LZ78 family, despite being heavily patent-encumbered. LZW improves on LZ78 in a similar way to LZSS; it removes redundant characters in the output and makes the output entirely out of pointers. It also includes every character in the dictionary before starting compression, and employs other tricks to improve compression such as encoding the last character of every new phrase as the first character of the next phrase. LZW is commonly found in the Graphics Interchange Format, as well as in the early specificiations of the ZIP format and other specialized applications. LZW is very fast, but achieves poor compression compared to most newer algorithms and some algorithms are both faster and achieve better compression.

6.8 LZC

LZC, or Lempel-Ziv Compress is a slight modification to the LZW algorithm used in the UNIX compress utility. The main difference between LZC and LZW is that LZC monitors the compression ratio of the output. Once the ratio crosses a certain threshold, the dictionary is discarded and rebuilt.

6.9 LZT

Lempel-Ziv Tischer is a modification of LZC that, when the dictionary is full, deletes the least recently used phrase and replaces it with a new entry. There are some other incremental improvements, but neither LZC nor LZT is commonly used today.

6.10 LZMW

Invented in 1984 by Victor Miller and Mark Wegman, the LZMW algorithm is quite similar to LZT in that it employs the least recently used phrase substitution strategy. However, rather than joining together similar entries in the dictionary, LZMW joins together the last two phrases encoded and stores the result as a new entry. As a result, the size of the dictionary can expand quite rapidly and LRUs must be discarded more frequently. LZMW generally achieves better compression than LZT, however it is yet another algorithm that does not see much modern use.

6.11 LZAP

LZAP was created in 1988 by James Storer as a modification to the LZMW algorithm. The AP stands for "all prefixes" in that rather than storing a single phrase in the dictionary each iteration, the dictionary stores every permutation[7].

6.12 LZWL

LZWL is a modification to the LZW algorithm created in 2006 that works with syllables rather than than single characters. LZWL is designed to work better with certain datasets with many commonly occuring syllables such as XML data. This type of algorithm is usually used with a preprocessor that decomposes the input data into syllables [8].

6.13 LZJ

Matti Jakobsson published the LZJ algorithm in 1985 and it is one of the only LZ78 algorithms that deviates from LZW. The algorithm works by storing every unique string in the already processed input up to an arbitrary maximum length in the dictionary and assigning codes to each. When the dictionary is full, all entries that occurred only once are removed[8].

VII. LEMPEL–ZIV–WELCH (LZW)

LZW is a universal lossless data compression algorithm [10] created by Abraham Lempel, Jacob Ziv,and Terry Welch. It was published by Welch in 1984 as an improved implementation of the LZ78 algorithm published by Lempel and Ziv in 1978. The algorithm is simple to implement, and has the potential for very high throughput in hardware implementations LZW is referred to as a *dictionary-based encoding algorithm.* The algorithm builds a *data dictionary* (also called a *translation table* or *string table*) of data occurring in an uncompressed data stream. Patterns of data (*substrings*) are identified in the data stream and are matched to entries in the dictionary. If the substring is not present in the dictionary, a code phrase is created based on the data content of the substring, and it is stored in the dictionary. The phrase is then written to the compressed output stream[2].

VIII. CONCLUSION

Lossless Data Compression Algorithms are help to compress a huge amount of data as to carry from one place to another or a in a storage format. Compression techniques are improved the efficiency compression on text data. Lempel-Ziv Algorithm is best of these Algorithms.

REFERENCES

[1] Haroon A, Mohammed A. Data Compression Techniques on Text Files: A Comparison Study International Journal of Computer Applications (0975 –8887) Volume 26– No.5, July 2011.
[2] P.Yellamma Dr.Narasimham Challa. Performance Analysis Of Different Data Compression Techniques On Text File October-2012.
[3] Mamta Sharma. Compression Using HuffmanCoding IJCSNS International Journal of ComputerScience and Network Security, VOL.10 No.5, May 2010
[4] Senthil Shanmugasundaram, Robert Lourdusamy. A Comparative Study Of Text Compression Algorithm International Journal of Wisdom Based Computing, Vol.1 (3),
[5] Draft Lecture Note compression Algorithms: Huffman and Lempel-Ziv-Welch (Lzw) Last Update: February 13, 2012.
[6] S.R. Kodituwakku. U. S.Amarasinghe Comparison Of Lossless Data Compression Algorithms For Text Data.
[7] Bell, T., Witten, I. Cleary, J.Modeling for Text Compression ACM Computing Surveys, Vol. 21, No. 4 1989.
[8] http://www.ieeeghn.org/wiki/index.php/Historyof Lossless Data Compression Algorithms
[9] Data compression Wikipedia.