International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 1 Issue 8, October - 2012

Performance Analysis Of Different Data Compression Techniques On Text File

P.Yellamma
Amrita sai institute of science and Technology, India

Dr.Narasimham Challa
Amrita sai institute of science and Technology India.

Abstract

Data Compression is the science and art of representing information in a compact form. Compression is the process of coding that will effectively reduce the total number of bits needed to represent certain information.Data compression has been one of the critical enabling technologies for the ongoing digital multimedia revolution .data compression also called as source coding or bit-rate reduction. 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 and calculating the entropy on English text files: Shanon-Fano coding, Huffman Encoding, Run-Length Encoding (RLE), Lempel-Ziv-Welch (LZW).

Keywords: lossless data compression, lossy data compression, Entropy, Shannon-Fano coding, Huffman encoding, RLE, LZW.

1. Introduction

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 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 lossy compression:

1.1.1. **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. [13]

1.1.2. **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 [5].

Compression ratio=B_1/B_0*100%.

B_0=no. of bits before compression.
B_1= no. of bits after compression

2. 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 a longer encoding vector. Shannon-Fano coding[1].

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]. A simple example will be used to illustrate the algorithm.

Example: Alice_has_sent_a_message_to_bob.

2.1. Algorithm

Encoding for the Shannon-fano algorithm: It follows a top-down approach [9].

Step1: Create table providing frequencies / counts

Step2: sort symbols according to their frequencies/ Probabilities in descending order.(table1)

Step3: Recursively divided into two parts, each with
approx (binary) same number of counts.

Step4: Add a binary 0 to the code words of the upper part and a binary 1 to the lower part.

Step5: Search for the next part containing more than two symbols, repeat the step3 and step4.

Step6: Coding of the origination data according to code words in the table2.

Step7:Create the coding tree(figure1).

Step8: Transmit Codes instead of Tokens

"**Table1**.Symbol table frequencies in descending order."

Symbol	_	a	e	s	t	o	b	l	l	c	h	n	m	g	.
Count	6	4	4	4	2	2	2	1	1	1	1	1	1	1	1

"**Table2**. Shannon-fano code words table."

symbol	count	step1	step2	step3	step4	step5	code
_	6	0	0	0			3
a	4	0	0	1			3
e	4	0	1	0			3
s	4	0	1	1			3
t	2	1	0	0	0		4
o	2	1	0	0	1		4
b	2	1	0	1	0		4
l	1	1	0	1	1		4
l	1	1	1	0	0	0	5
c	1	1	1	0	0	1	5
h	1	1	1	0	1	0	5
n	1	1	1	0	1	1	5
m	1	1	1	1	0	0	5
g	1	1	1	1	0	1	5
.	1	1	1	1	1		4

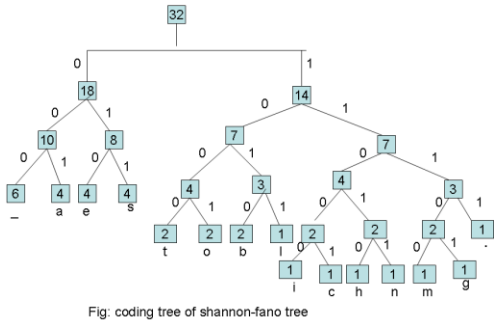


Fig: coding tree of shannon-fano tree

"**Figure1**.Shannon-fano coding tree."

"**Table3**.Result of performing Shannon-fano."

Symbol	Count	Codeword	# of bits used
_	6	000	18
a	4	001	12
e	4	010	12
s	4	011	12
t	2	1000	8
o	2	1001	8
b	2	1010	8
l	1	1011	4
i	1	11000	5
c	1	11001	5
h	1	11010	5
n	1	11011	5
m	1	11100	5
g	1	11101	5
.	1	1111	4

Table: Result of Performing shanon-fano

Total number of used bits=116.
Before compression (B_0) =32*8=256bits.
After compression (B_1) =116bits.
Compression ratio=B_1/ B_0 *100%
=116/256*100%=45.3% from the original size.
It means that it saves 54.7% in space[1].

Generally, Shannon-Fano coding does not guarantee that an optimal code is generated. Shannon – Fano algorithm is more efficient when the probabilities are closer to inverses of powers of 2.

3. Huffman Encoding

The Huffman coding algorithm [4] is named after its inventor, David Huffman, who developed this algorithm as a student in a class on information theory at MIT in1950. It is a more successful method used for text compression. It is a compression algorithm used for loss-less data compression.

A more efficient approach is to use a variable-length representation, where each character can have a different number of bits to represent it. More specifically we first analyze the frequency of each character in the text, and then we create a binary tree (called Huffman tree) giving a shorter bit representation to the most used characters, so that they can be reached faster[3].

3.1. Algorithm

Step1: Compute the probability of each character.

Step2: Sort the set of data in ASCENDING order.

Step3: Create a new node where the left child is the lowest in the sorted list and the right is the second lowest in the sorted list.

Step4: Chop-off those elements in the sorted list as they are now part of one node and add the probabilities. The result is the probabilities for the new node.

Step5: Perform insertion sort on the list with the new node.

Step6: Repeat steps 3, 4, 5 until, only have one node left.

Step7: Calculate Entropy

Example: Alice_has_sent_a_message_to_bob.

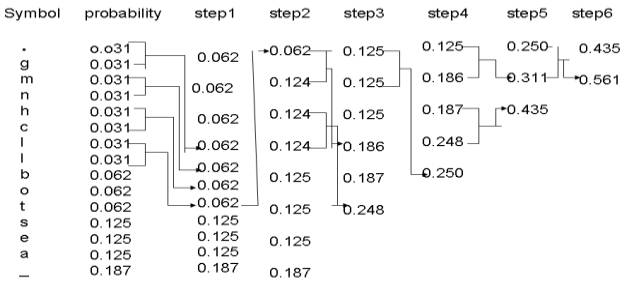
"**Table1**.Symbol table frequencies in Ascending order."

Symbol	.	g	m	n	h	c	l	l	b	o	t	s	e	a	_
Count	1	1	1	1	1	1	1	1	2	2	2	4	4	4	6

It is based on building a full binary tree for the different symbols that are in the original file after calculating the probability for each symbol and put them in ascending order. After that, we derive the code words for each symbol from the binary tree, giving short code words for symbols with large probabilities and longer code words for symbols with small probabilities.

By applying Huffman algorithm on the given example. We get the probability table [2].

"**Table5.** Ascending probabilities for symbols".



Fig; Huffman coding of sorted probability data

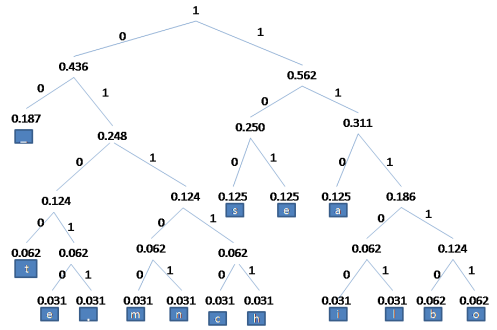


Fig: huffman encoding tree

"**Figure 2:** Huffman encoding tree".

Huffman Method of Code Generation and average code length per character computation is calculating in this example [7].

"**Table 6:** Huffman code words symbols".

Symbol	no.of count	probability(pi)	log(1/pi)	codeword	no.of.bits(Li)	codeword length	#bits of used
_	6	0.187	o.726	00	2	0.374	12
a	4	0.125	0.903	110	3	0.375	12
e	4	0.125	0.903	101	3	0.375	12
s	4	0.125	0.903	100	3	0.375	12
t	2	0.062	1.204	0100	4	0.248	8
o	2	0.062	1.204	11111	5	0.31	10
b	2	0.062	1.204	11110	5	0.31	10
l	1	0.031	1.505	11101	5	0.155	5
l	1	0.031	1.505	11100	5	0.155	5
c	1	0.031	1.505	01110	5	0.155	5
h	1	0.031	1.505	01111	5	0.155	5
n	1	0.031	1.505	01101	5	0.155	5
m	1	0.031	1.505	01100	5	0.155	5
g	1	0.031	1.505	01011	5	0.155	5
.	1	0.031	1.505	01010	5	0.155	5

Average code word length=$\sum$ pi*Li Average code word length=3.607

Entropy=ΣPi*log(1/pi)
= [(0.187*2.4)+(3*(0.125*3))+(3*(0.062*4))+(8*(0.031*5))
=0.45+1.125+0.744+1.24=3.559 bits/symbol.

Average code word length=entropy=~3.6 bits/symbols. Huffman weighted path in this example is:

Weighted path=$\sum$ (no. of counts*no. of code words).[5]

Weighted path=[(6*2)+(4*3)+(4*3)+(4*3)+(2*4)+(2*5)+(2*5)+(1*5)+(1*5)+(1*5)+(1*5)+(1*5)+(1*5)+(1*5)+(1*5)]
=12+12+12+12+8+10+10+5+5+5+5+5+5+5+5=116bits

Compression ratio=B_1/B_0*100%=116/256*100%
=45.3% from the original size. it means that it saves 54.7% in space.

4. Run-length encoding (RLE)

Run Length Encoding (RLE) is a simple and 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.

4.1. General Principle of RLE [16]

Instead of the original data so-called runs will be stored. In the general form a run is a sequence of a certain length containing only one symbol. The length of the sequence is called run count and the symbol run value

4.2. Algorithm.

step1: Pick the character from source string.
step2: Append the picked character to the destination string.
step3: Count the number of subsequent occurrences of the picked character and append the count to destination string.
step4: Pick the next character and repeat steps 2, 3 and 4 if end of string is NOT reached.

Example: Alice_has_sent_a_message_to_bob.

"**Table7:** RLE data compression hypothetical scan line".

Run	_ _ _ _ _ _	aaaa	eeee	ssss	tt	oo	bb	l	i	c	h	n	m	g	.
Run length	6	4	4	4	2	2	2	1	1	1	1	1	1	1	1
Run value	_	a	e	s	t	o	b	l	i	c	h	n	m	g	.

original data: alice_has_sent_a_message_to_bob.

RLEcoded: 6_4a4e4s2t2o2b1l1i1c1h1n1m1g1.

If we apply the run-length encoding (RLE) data compression algorithm to the above hypothetical scan line, we get the following:

6_4a4e4s2t2o2b1l1i1c1h1n1m1g1.

Compression ratio=$B_{1/}$ B_0 *100%

=30/32*100%=93.7%from the original size. It means that it saves 6.3% in space only. Does not work well for English text however, is almost always a part of a larger compression system.

5. 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 [6].

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 [1].

Code table compression is the basis of the popular LWZ compression method. Encoding occurs by identifying sequences of bytes in the original file that exist in the code table. The 12 bit code representing the sequence is placed in the compressed file instead of the sequence. The first 256 entries in the table correspond to the single byte values, 0 to 255, while the remaining entries correspond to sequences of bytes. The LZW algorithm is an efficient way of generating the code table based on the particular data being compressed.

5.1. LZW encoding algorithm:

Encoding input consists of the following steps:

Step 1. Initialize dictionary to contain one entry for each byte. Initialize the encoded string with the first byte of the input stream.
Step 2. Read the next byte from the input stream.
Step 3. If the byte is an EOF goto step 6.
Step 4. If concatenating the byte to the encoded string produces a string that is in the dictionary:

- Concatenate the the byte to the encoded string.
- Go to step 2.

Step 5. If concatenating the byte to the encoded string produces a string that is not in the dictionary:

- Add the new sting to the dictionary.
- Write the code for the encoded string to the output stream.
- Set the encoded string equal to the new byte.
- Go to step 2.

Step 6. Write out code for encoded string and exit.

Example: alice_has_sent_a_message_to_bo**.**

The LZW algorithm stores in a “dictionary”. The first 255 entries are used to contain the values for individual.

S=alice_has_sent_a_message_to_bob.			
Character	code output	new code	new string
al	a	256	al
l	l	257	li
c	l	258	ic
e	c	259	ce
_	e	260	e_
h	_	261	_h
a	h	262	ha
s	a	263	as
_	s	264	s_
s	_	265	_s
e	s	266	se
n	e	267	en
t	n	268	nt
_	t	269	t_
a	_	270	_a
_	a	271	a_
m	_	272	_m
e	m	273	me
s	e	274	es
s	s	275	ss
a	s	276	sa
g	a	277	ag
e	g	278	ge

e_ but this has already been included in the dictionary in entry 260.this means we now set up the three-character string "e_t".

Chracter	code output	new code	new string
_t	260	279	e_t
o	t	280	to
_	o	281	o_
b	_	282	_b
o	b	283	bo
b	o	284	ob
.	b	285	b.
EOF	.	286	.

By using LZW compression algorithm our examples is not suitable. Generally LZW is suitable for the images. In this example Compression ratio=31/32*100=96.8% from the original size. it means that it saves 3.2% in space only or storage of new file. [4]

6. Analysis

In this section, tests are made on the four compression techniques on same text file. The results are tabulated and analyzed in order to reach to the best technique, advantage and disadvantage for each one, and when each one is best to use. First compare the Shannon-fano and Huffman coding, the compression ratio is almost same. By using these compression algorithms it saves the 54.7%space.Second compare the Run length encoding and Lempel-ziv-welch algorithms of the compression ratio is low as compare with the Huffman and Shannon-fano algorithms. According to my observation Huffman encoding algorithm is best result for the text files.

7. Conclusion

Data compression is most consideration thing of the recent world. We have to compress a huge amount of data so as to carry from one place to other or in a storage format. These proposed compression technique are improved the efficiency of compression on text. Huffman encoding Algorithm is suitable for the given text.

8. 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 201.

[2]. Mohammed Al-laham & Ibrahiem M. M. El Emary "Comparative Study between Various Algorithms of Data Compression Techniques" Proceedings of the World Congress on Engineering and Computer Science 2007WCECS 2007, October 24-26, 2007, San Francisco, USA.

[3]. Mamta Sharma "Compression Using Huffman Coding". IJCSNS International Journal of Computer Science and Network Security, VOL.10 No.5, May 2010

[4].Senthil Shanmugasundaram, Robert Lourdusamy "A Comparative Study Of Text Compression Algorithms". International Journal of Wisdom Based Computing, Vol. 1 (3), December 2011

[5]. 2shared.document Chapter 7"Lossless Compression Algorithms".ppt.

[6]. Draft Lecture Notes"compression Algorithms: Huffman and Lempel-Ziv-Welch (Lzw)". Last Update: February 13, 2012.

[7].Double Compression Of Test Data Using Huffman Code

[8].Dave Marshall " Lossless Compression Algorithms".

[9]. Roger seeck " Shannon-Fano Coding"

[10].Roger seeck"General Principle of RLE"

[11].S.Aarthi, D. Muralidharan, P. Swaminathandouble "Compression Of Test Data Using Huffman Code" Journal Of Theoretical And Applied Information Technology 15 May 2012. Vol. 39 No.2.