International Journal of Applied Engineering Research ISSN 0973-4562 Volume 13, Number 6 (2018) pp. 3296-3414
© Research India Publications. http://www.ripublication.com

The Pillars of Lossless Compression Algorithms a Road Map and Genealogy Tree

Evon Abu-Taieh, PhD

Information System Technology Faculty, The University of Jordan, Aqaba, Jordan.

Abstract

This paper presents the pillars of lossless compression algorithms, methods and techniques. The paper counted more than 40 compression algorithms. Although each algorithm is an independent in its own right, still; these algorithms interrelate genealogically and chronologically. The paper then presents the genealogy tree suggested by researcher. The tree shows the interrelationships between the 40 algorithms. Also, the tree showed the chronological order the algorithms came to life. The time relation shows the cooperation among the scientific society and how the amended each other's work. The paper presents the 12 pillars researched in this paper, and a comparison table is to be developed. Furthermore, novice users of the algorithms and researchers can find a reference where they can use this research as spring board for future work. The researchers, wanted to accompany each algorithm with a classical example to make understanding easier and recalling the algorithm much faster with an example rather than an algorithm.

Keywords: Compression algorithms, Haar, Huffman, LZW, bizip2, RLE, MTF, SHANNON, Coding and Information Theory.

INTRODUCTION

This paper presents the pillars of lossless compression algorithms, methods and techniques. In compression algorithms, there are two types lossy and lossless. The lossless compression algorithms keep the file in its original state before and after compression. On the other hand, lossy compression algorithm the file loses some of its quality after the compression process. The paper counted more than 40 compression algorithm: Arithmetic Coding - SHANNON (1948), Huffman coding (1952), FANO (1949), Run Length Encoding (1967), Peter's Version (1963), Enumerative Coding (1973), LIFO(1976), FiFO Pasco(1976), LZ77(1977), Move To Front (MTF) Transform (1980), LZ78(1978), LZR (1981), LZSS (1982), LZJ(1985), LZC(1985), LZZMW(1985), LZB(1987), LZH(1987), LZT(1987), LZAP (1988), LZRW (1991)), DEFLATE(1993), LZS(1994), LZP(1995), LZX(1995), LZO(1996), LZMA, LZJB, LZWL(2006), LZMA2(2009), LZ4(2011), Burrows- Wheeler Transform (1994), Haar (1910), Wavelet tree(2003), Stream(1979), P-Based FIFO(1981), Delta Encoding, Rice & Golomb Coding (1966,1979), Run-Length Golomb-Rice (RLGR) (2007), Tunstall coding (1967). Although each algorithm is an independent in its own right, still; these algorithms interrelate genealogically and chronologically. Hence, the major stubs in the developed tree of the compression algorithms are 12. The tree is presented in the last section of the paper after presenting the 12 main compression algorithms each with a practical example.

The paper first introduces Shannon–Fano code showing its relation to Shannon (1948), Huffman coding (1952), FANO (1949), Run Length Encoding (1967), Peter's Version (1963), Enumerative Coding (1973), LIFO (1976), FiFO Pasco (1976), Stream (1979), P-Based FIFO (1981). Two examples are to be presented one for Shannon-Fano Code and the other is for Arithmetic Coding. Next, Huffman code is to be presented with simulation example and algorithm. The third is Lempel-Ziv-Welch (LZW) Algorithm which hatched more than 24 algorithms. The LZW family is to be presented and a working example is given. The fourth is Run Length Encoding (RLE) which essentially codes using frequency and position. The fifth is Burrows-Wheeler Transform. The sixth is Move-to-Front (MTF) Transform. The seventh is Haar and the eighth is wavelet tree. The ninth is the Delta Encoding, and the tenth is Rice & Golomb Coding. The eleventh is Tunstall coding. The twelfth is a hybrid example bzip2 which is a mix of many algorithms & DEFLATE algorithm which is also a mix of three algorithms. The last example of the hybrid is Run-Length Golomb-Rice (RLGR).

The paper will present the genealogy tree suggested by the authors. The tree will show the interrelationships between the 40 algorithms. Also, the tree will show the chronological order the algorithms came to life. The time relation shows the cooperation among the scientific society and how they amended each other's work. Furthermore, novice users of the algorithms and researchers can find a reference where they can use this research as spring board for future work. The researcher, wanted to accompany each algorithm with a classical example to make understanding easier and recalling the algorithm much faster with an example rather than an algorithm.

SHANNON–FANO CODE

In 1948 Shannon proposed a technique in an article titled "A mathematical Theory of Communication" (Shannon, 1948) and in 1949 Fano proposed a method that was published in a technical report titled "The transmission of information" (Fano, 1949). Hence, a term was dubbed as "Shannon-Fano Code". Peter Elias improved the algorithm in unpublished work that was later described by (Abramson, 1963). In 1973 (Cover, 1973) published another version named Enumerative Coding. In 1976 (Rissanen, 1976) introduced Last in First Out (LIFO) version of the algorithm. In 1976 Pasco (Pasco, 1976) introduced the FIFO version of the algorithm. In 1979 "stream" code was discovered by Rubin (1979) as an improvement of Pasco's work. Martin (1979) and Jones (1981) developed P-based FIFO arithmetic codes (Jones, 1981) and (Martin, 1979). The algorithm Shannon-Fano code is explained in the next section by showing the algorithm and an example is given in section A and section B.

A) Shannon–Fano Algorithm

Shannon-Fano algorithm is presented in the five steps below; the algorithm is recursive in nature and easily coded and understood.

1. Scan text to be compressed and count the occurrence of all characters.
2. Sort the lists of symbols according to frequency, with the most frequently occurring symbols at the left and the least common at the right.
3. Divide the list into two parts, with the total frequency counts of the left part being as close to the total of the right as possible.
4. The left part of the list is assigned the binary digit 0, and the right part is assigned the digit 1. This means that the codes for the symbols in the first part will all start with 0, and the codes in the second part will all start with 1.
5. Recursively apply the steps 3 and 4 to each of the two halves, subdividing groups and adding bits to the codes until each symbol has become a corresponding code leaf on the tree.

B) Classical Example

This is a classical of example of Shannon–Fano algorithm (Shannon, 1948), consider the following text

ABRACADABRA ABRACADABRA.

The first step is "Scan text to be compressed and count the occurrence of all characters". The seven symbols which can be coded have the following, table 1, is frequency table:

Table 1. Frequency table of the scanning phase of the algorithm**.**

Character	**Frequency**	**Probabilities**	**code**
A	10	0.416667	00
R	4	0.166667	01
B	4	0.166667	100
C	2	0.083333	101
D	2	0.083333	110
SP	1	0.041667	1110
period	1	0.041667	1111

The second step according to the algorithm is show in graph below and marked as step #1. As shown, the priority list and the character are sorted according to frequency in descending order. In figure 1 step#2, the list is broken into two groups as mentioned in the algorithm above, with the total frequency counts of the left part being as close to the total of the right as possible. Hence the first sub list composed of letter A and letter R with a total frequency is 14, while the second sub list is composed of B, C, D, SP, Period with total frequency is 10.

In step #3 each sub list is further divided into two sub lists. Each list must adhere to algorithm condition (the total frequency counts of the left part being as close to the total of the right as possible). The first list is further divided into two leaves with frequency 10 and 4 respectively.

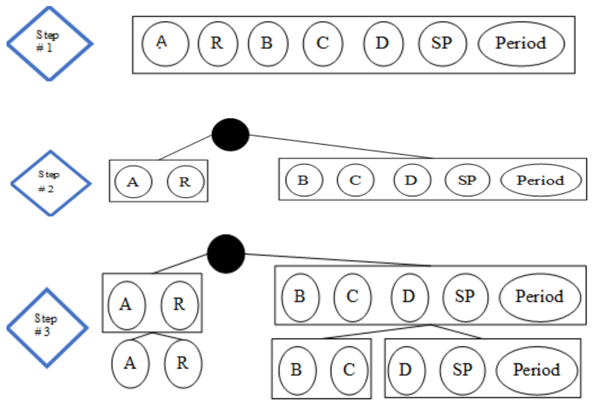


Figure 1: Shannon–Fano algorithm tree of the example steps 1 & 2 & 3.

The second list is divided into two lists, the first one is letters B & C with total frequency 6 and the second list is D, SP, Period with total frequency of 4. In step#4, each sub list is further divided in accordance to previously mention condition, see figure 2. The B &C list is divided into two leaves; the second list (D, SP, and Period) is further divided into leaf D with frequency 2 and list SP & Period with total frequency 2. In the step #5, the last sub list SP & Period is divided into two leaves as show in figure 3. The product is a binary tree, the tree branches are labeled according to algorithm (left branch is labeled 0, and right branch is labeled 1).

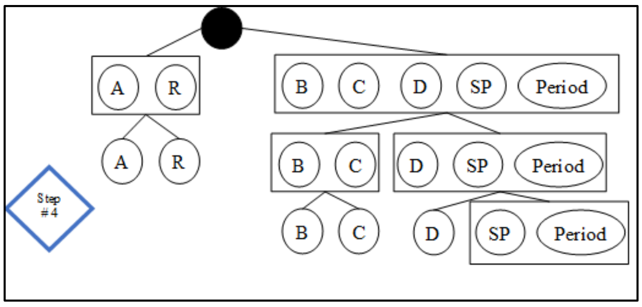


Figure 2. Operations of step #4.

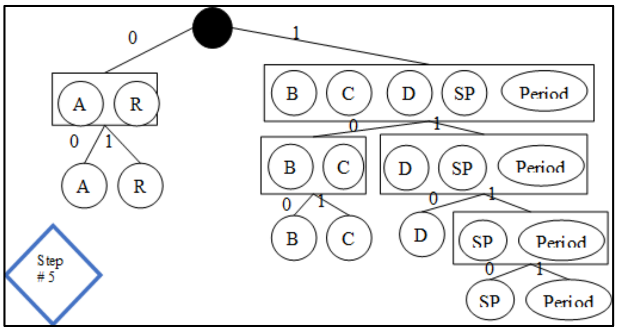


Figure 3. Shannon–Fano algorithm tree of the example steps 5.

The text length is 24 letters usually coded using 8-bit fixed length which is 24*8 = 192 bit. Using Shannon–Fano code the 24 letters were coded using 60 bits as shown in table 2. Hence, the savings are 1- 60/192=.6875 which is almost 68.75%.

Table 2. Table showing the savings using Shannon-Fano algorithm.

Character	Frequency	Code length	Frequency X Code length
A	10	2	10*2
R	4	2	4*2
B	4	3	4*3
C	2	3	2*3
D	2	3	2*3
SP	1	4	1*4
period	1	4	1*4
Total bit			**60**

C) Example of Arithmetic Coding

As an example of arithmetic coding, the following example is presented. Code the word "**COMCA**" (Langdon, 1984)

Symbol	Frequency	Probability	Interval
C	2	0.4	[0-0.4]
O	1	0.2	[0.4-0.6]
M	1	0.2	[0.6-0.8]
A	1	0.2	[0.8-1]
TOTAL	**5**	**1**	

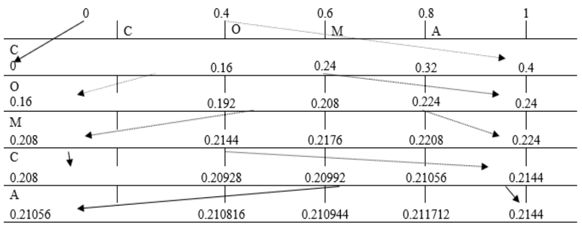


Figure 4. Step by step Arithmetic Coding

Since the letter, **C** is the first of the required word we start by expanding the probability line number see figure 4.

(0.4-0) *0.4+0=0.16

(0.4-0) *0.2+0.16=0.24

(0.4-0) *0.2+0.24=.32

The second symbol is **O** so we start expanding the limits of the **O** [0.16, 0.24]

(0.24-0.16) *0.4+0.16=0.192

(0.32-0.24) *0.2+0.192=.208

(0.4-0.32) *0.2+0.208=0.224

Then we move on to letter **M,** and start limit with 0.208

(0.208-0.192) *0.4+0.208=0.2144

(0.224-0.208) *0.2+0.2144=0.2176

(0.24-0.224) *0.2+0.2176=0.2208

Then letter **C** and start limit with 0.208

(0.2144 - 0.208) * 0.4 + 0.208 = 0.20928

(0.2208 - 0.2176) * 0.2 + 0.2092 8= 0.20992

(0.224 - 0.2208) * 0.2 + 0.20992 = 0.21056

Next is the letter **A**, see figure 4, the Lower limit is 0.21056

(0.20992-0.20928) * 0.4 + 0.21056=0.210816

(0.21056-0.20992) * 0.2 + 0.210816=0.210944

(0.2144-0.21056) *0.2 + 0.210944=0.211712

The coded word for COMCA is 0.2144, see figure 4. To decode the word back we decode 0.2144 as follows:

CR=upper limit - Lower limit of current interval……..(1)

ES=ES - lower limit of current interval………………(2)

ES=ES/CR…………………..………………………….(3)

Since the coded word is 0.2144 hence the interval [0-0.4] applies which belongs to letter **C**.

CR = 0.4 - 0.0=0.4

ES = 0.2144 - 0.0

ES = 0.2144/0.4=0.536

Since ES is in interval [0.4-0.6] the letter is **O**

CR = 0.6-0.4 = 0.2

ES = 0.536-0.4 = 0.136

ES = 0.136/0.2 = 0.68

Since ES is in interval [0.6-0.8] then the decoded letter is **M**

CR=0.8-0.6-0.2

ES=0.68-0.6=0.08

ES=0.08/0.2=0.4

Since ES is in interval [0.0-0.4] then the decoded letter is **C**

CR=04-0.0=0.4

ES=0.4-0.0=0.4

ES=0.4/0.4=1

Since ES is in interval [0.8-1.0] then the decoded letter is **A**

HUFFMAN CODE

Huffman coding is a technique used to compress files for transmission; it is a variable-length encoding/compression scheme. Proposed by Dr. David A. Huffman in 1952 in a paper titled "A Method for the Construction of Minimum Redundancy Codes" (Huffman, 1952). The technique uses statistical coding (most frequently used symbols have shorter code words). The technique uses several data structures: queues, trees, etc. In the next section the algorithm is presented and explained using an example.

A) Huffman Code Algorithm

Huffman Code algorithm is presented in the six steps below, the algorithm will produce the tree shown in figure 5, and the algorithm is easily coded and understood.

1. Scan text to be compressed and count the occurrence of all characters.
2. Create priority list: Sort or characters based on their frequency in text.
3. Build Huffman code tree based on prioritized list.
4. Perform a traversal of tree to determine all code words.
5. Scan text again and create new file using the Huffman codes.
6. send coded file and the tree for decoding

B) Classical Example

To fully understand Huffman code, an example is presented here. Consider the following text: "ABRACADABRA ABRACADABRA." The first step is "Scan text to be compressed and count the occurrence of all characters", the following table of frequency will result from applying the first step of the algorithm:

Character	Frequency
A	10
R	4
B	4
C	2
D	2
SP	1
period	1

The second step according to the algorithm is shown in figure 5 below and marked as step #1. As shown, the priority list and the character are sorted according to frequency in ascending order.

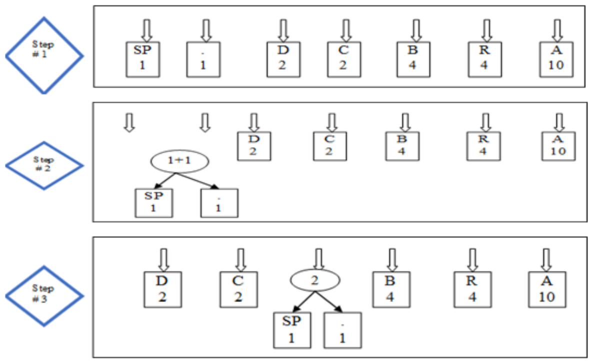


Figure 5. Steps 1, 2 and three explained graphically.

In step#2, choose the lowest frequency nodes and co-join the two with one node and add their frequency to be the frequency of the new node. In this case, the space and period each has frequency of 1, the new node has the frequency 2, see figure 5. In step #3, insert the new node before the nodes of D & C, both have frequency of 2, figure 5.

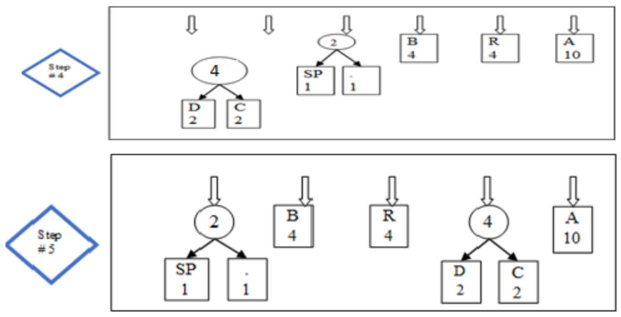


Figure 6. Operations carried out in step #4 D & C, and step #5.

In step #4, repeat what happened in step #2, picture the first two nodes with lowest frequency in the queue. Co-join the two node C &D in one node and the new node frequency is 4, see figure 6 step # 4. Step #5, insert the new node in the queue before "R" and after node "A", see figure 6.

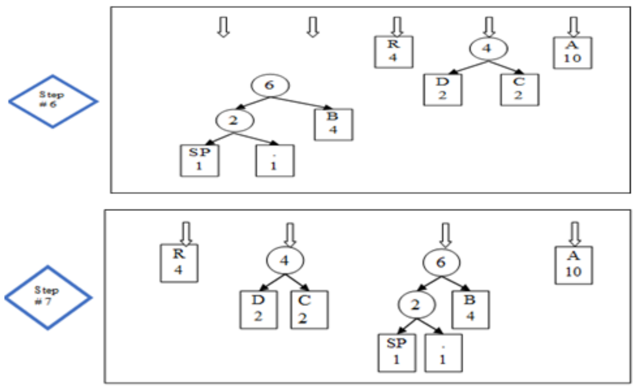


Figure 7. Operations of step # 6 and step # 7.

Step #6, co-join the SP and period with count=2 with the B count=4 the sum of the count is 6 now, see figure 7. In preparation for step #7. In step#7 The sub-tree of count=6 is put in queue between the A of count=10 and the subtree of count=6, see figure 7.

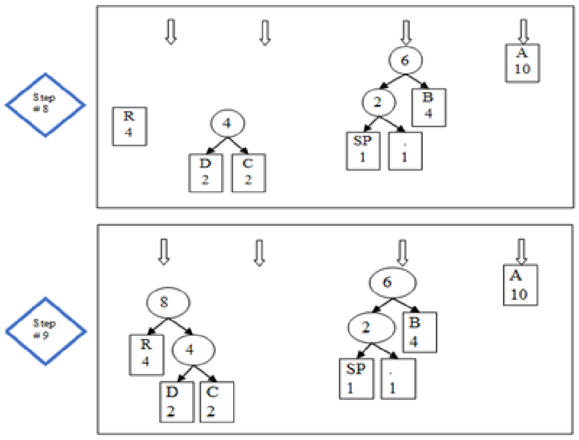


Figure 8. Operations of step # 8 and step #9.

In step # 8, co-join the r with count =4 and the D-C subtree with count=4, the total count is shown in step# 9 figure 8. In step# 9, new count is set for the R and the sub-tree (D-C). the new count is 8.

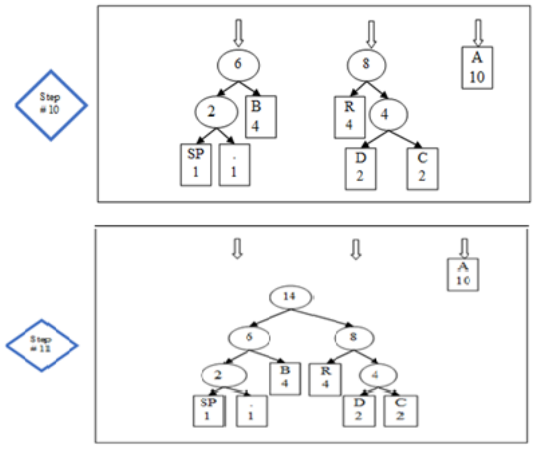


Figure 9. Operations of step # 10 and step #11.

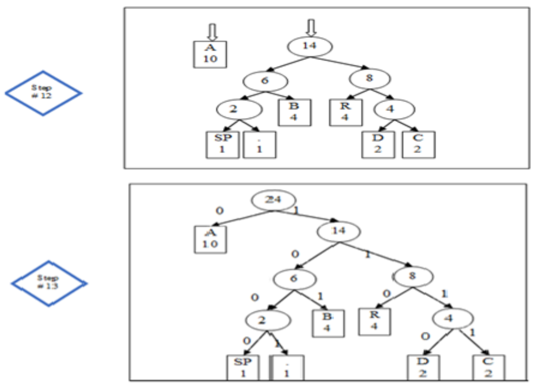


Figure 10. The operations in step # 12step #13.

In step # 9 the new sub-tree with count=8 is now to be placed after the A of count=10, as seen in figure 8. The Sub tree is put in the queue after the A with count=8, while count of the A is greater than 8, it is 10, as seen in figure 9.

In step #11, the two sub-trees are co-joined: the first with count=6 and the second subtree with count=8. The total is 14 as seen in in figure 9. The step # 12, the new sub-tree of count=14, is but in the queue before the A of count=10. As seen in figure 10. In step#13, co-join the two sub-trees the leaf A with count =10 and the sub-tree with count=14,

After going through all 13 steps, the final tree appears as seen in figure 11. Now each letter is coded according to the binary branches value. The A is coded to be 0, the SP is coded as 1000, the period (.) is coded as 1001, the B with more frequency is coded as 101, the R with frequency 4 is coded as 110, the D with frequency is 2 is coded as 111, the C with frequency 2 is coded as 1111. Now each letter in the coded word is replace by its code. The A is replaced by 0s, the B with 101, the R with 110, the D with 111, The C with 1111, the period with 1001 and the space with 1000. The process is seen in figure 11.

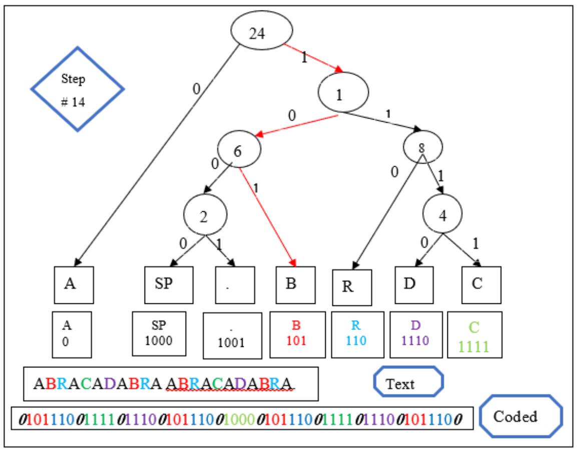


Figure 11. Huffman coding, the final tree of the code.

One can conclude that the letter with highest frequency is replaced with shortest code. For example, A with highest frequency (10) is coded by one bit which is (0). While the letter with least frequency like the period and SP are, both coded with 4 bits 1001and 1000 respectively. Hence the intended word "ABRACADABRA ABRACADABRA" was coded with 58 bits (010111001111011100101110010000101110011110111001011100).

The text length is 24 letters usually coded using 8-bit fixed length which is 24*8 = 192 bit. Using Huffman algorithm, the 24 letters were coded with only 58 bits instead of 192 bits as seen in table 3. Hence, the savings are 1- 58/192=.697 which means 69.7% savings.

Table 3 Showing the savings when Huffman is used.

Character	Frequency	Code length	Frequency X Code length
A	10	1	10*1
R	4	3	4*3
B	4	3	4*3
C	2	4	2*4
D	2	4	2*4
SP	1	4	1*4
period	1	4	1*4
Total bits			**58**

Huffman code reduced what could have been coded in 192 bits to 58 bits which is 69.7% saving in storage. Still, in order to send such coded word, the whole tree must be sent to be used in the decoding process. Furthermore, what Huffman saved in compression countered it in the sending process. The computational complexity of Huffman code is O (n log n)., where n is the number of symbols used in the text message.

LEMPEL-ZIV-WELCH (LZW) ALGORITHM

Historically LZ was first developed in 1977 by Jacob Ziv and Abraham Lempel, and was name LZ77. In 1978, both Ziv and Lemple, developed the algorithm and named it LZ78. Many LZ based algorithms were developed over the years: in 1981 LZR was developed, in 1982 LZSS Lempel-Ziv-Storer-Szymanski by James Storer and Thomas Szymanski, in1984 LZW, in 1985 LZJ, LZC, and LZZMW; in 1987 LZB, LZH, LZT. In 1988 LZAP was developed, in 1991 LZRW, in 1993 DEFLATE, in 1994 LZS, in 1995 LZP, & LZX, in 1996 LZO, in 1998 LZMA & LZJB, in 2006 LZWL in 2009 LZMA2 and in 2011 LZ4.

LZW was developed in 1984; Terry Welch came on board with both authors and the three developed LZW. LZW is used in GIF file format. The compression algorithm is in figure 12 and the decompression algorithm is in figure 13.

```
set w = NIL
loop
read a character k
if wk exists in the dictionary
 w = wk
else
output the code for w
add wk to the dictionary
 w = k
endloop
```

Figure 12. LZW Compression algorithm.

```
read a character k
output k
w = k
loop
    read a character k
    entry = dictionary entry for k
    output entry
    add w + first char of entry to
the dictionary
    w = entry
endloop
```

Figure 13. LZW Decompression algorithm.

A) Classical Example LZW

To show the work of LZW a trace classical examples are shown in this section. Two words will be coded using LZW: "thisisthe" and "ABRACADABRABRABRA". The first example reduces a 10-letter word to 7 characters. The second example reduces 17 letters word to 12-character coded word. The first example we save 30% size the second saves even more 29.4%.

Compressing the following string (Cslearning, 2013) **thisisthe**. First, look at the current (first character) and the next character which is letter **(h)**, since letter **(t)** is in the alphabets output is set to t. Next, see if the two characters **(th)** are in the dictionary and since two characters **(th)** are not, then (**th)** are added to the dictionary. In addition, since (**th)** is not in our dictionary we give it a new value which is **(256)**. Second, the pointer moves to next letter which **(h)**, so the current letter (**h**) and the next letter is **(i)**, since **(hi)** is not in the dictionary a new value (**257)** is given to the two characters (**hi**). Third step, we move the pointer to the thirds letter (**i**). the next letter is **s**, and we see if **(is)** in the dictionary, since it is not we add **is** to dictionary and give it the value 258. The fourth step, the current letter is (**s**), the next letter is (**i**) since **(si)** not in dictionary we add it to dictionary and give it a value **(259)**. The fifth step, the current letter is (**i**), and the next letter (s), and (is) is in the dictionary so we look if (**ist)** is in the dictionary and since it is not we add it to the dictionary and we output (is) and add (ist) to dictionary and give it the value (260). Hence, we reduced the code by one letter. The sixth step, current letter is (t) and next letter is (h) and (th) is in dictionary so we output (th) is (256) and we add to dictionary (the) and give it the value (261). The seventh step, the current letter is e and the next is null we output (e) and add null to dictionary.

Step	Current		Next		Output		Add to Dictionary	
1	t	116	h	104	t	116	th	256
2	h	104	i	105	h	104	hi	257
3	i	105	s	115	i	105	is	258
4	s	115	i	105	s	115	si	259
5	i	105	s	115	"is" is in the dictionary but ist is not			
	is	258	t	116	is	258	ist	260
6	t	116	h	104	"th" is in dictionary			
	th	256	e	101	th	256	the	261
7	e	101	-		e	101	-	

Hence, we coded the text of length 10 characters into 7 characters, which means we compressed the text by almost 13%.

t	h	i	s	i s	t h	e
116	104	105	115	258	256	101

Using the same idea, try to co de "ABRACADABRABRABRA".

Current		Next		Output		Add to Dictionary	
A	41	B	42	A	41	AB	81
B	42	R	52	B	42	BR	82
R	52	A	41	R	52	RA	83
A	41	C	43	A	41	AC	84
C	43	A	41	C	43	CA	85
A	41	D	44	A	41	AD	86
D	44	A	41	D	44	DA	87
A	41	B	42	AB	81	ABR	88
R	52	A	41	RA	83	RAB	89
B	42	R	52	BR	82	BRA	8A
A	41	B	42	ABR	88	ABRA	8B
A	41	-		A	41	-	

The coded text will be as follows

A	B	R	A	C	A	D	A	B	R	A	B	R	A	B	R	A
41	42	52	41	43	41	44	81		83		82		88			41

Saving 5 out of 17 characters. Which means 1-12/17=.294=29.4%. The two examples show the work of LZW. Both showed the amount of compression conducted on 10 letter word and 17 letter word.

B) LZSS

LZSS is another flavor of LZ based compression algorithm. Developed by James Storer and Thomas Szymanski in 1982 (STORER & SZYMANSKI, 1982). In LZSS " 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" (STORER & SZYMANSKI, 1982) According to the same source the *NEXT* pointer was eliminated. Hence when compressing **"these theses"** the compressed is **"these(0,6)s"**, where the 0 is the index and 6 is the offset.

RUN LENGTH ENCODING (RLE)

RLE is lossless encoding schema. RLE is very simple and easy to use. RLE was used since 1967 to compress television signals. As can be seen in the next example: To encode the text "aaaabccabbbc" the answer is (a,4), (b,1), (c,2), (a,1), (b,3), (c,1). So basically, each letter is keeps its position and the frequency of the letter is counted both are put in (letter, frequency). The algorithm in RLE is shown in figure 14:

```
Counter=0
For i=1 to end of sequence
        If Sequence [i]=sequence[i+1] then
                Counter=counter+1
        Else
                Set pair= (Sequence [i], Counter)
                Counter=0
        Endif
```

Figure 14. RLE compression algorithm**.**

BURROWS- WHEELER TRANSFORM(BWT)

The Burrows-Wheeler Transform method was developed by two scientists: M. Burrows and D.J. Wheeler in 1994 in a technical report for Digital Equipment Corporation (Burrows & Wheeler, 1994). The following is an algorithm to clarify the process of BWT shown in figure 15. In other literature the suggested time complexity of the BWT is O (n log n) (Lippert, Mobarry, & Walenz, 2005).

To further explain the Burrows-Wheeler Transform method the following example is presented in the next section.

A) Classical Example of Burrows- Wheeler Transform

To code the word " ***abracadabra*** ". First put the word in a matrix with all possible rotations like shown in figure 16.a, in a matrix while shifting in each row by one character (space). Next, the matrix should look like the figure 16.b, the second row shifted last letter, will become first in the second row. Next, Sort the matrix according to column 1, then 2, then,3 etc. the result is the matrix figure 16.c.

```
To encode
Input: input sequence
Create BWT an nXn matrix, n is the length of input sequence.
For counter =1 to n do
Rotate input sequence (shift left by counter)
Insert rotated input sequence into row[counter]
Sort matrix BWT according to each column from 1..n, alphabetically
Find the row of the original input sequence in sorted-BWT, return the index (ptr)
Coded sequence is the last column, position of each letter and the index from previous step
To decode
Input: coded matrix with 2 columns: last column, letter position and ptr
First letter=coded matrix [ptr]. letter
Next=coded matrix.index[ptr]
Loop until done
        Next letter=coded matrix[next].letter
        Output next letter
        Next= coded matrix[next] . letter position
End loop
```

Figure 15. BURROWS- WHEELER TRANSFORM algorithm.

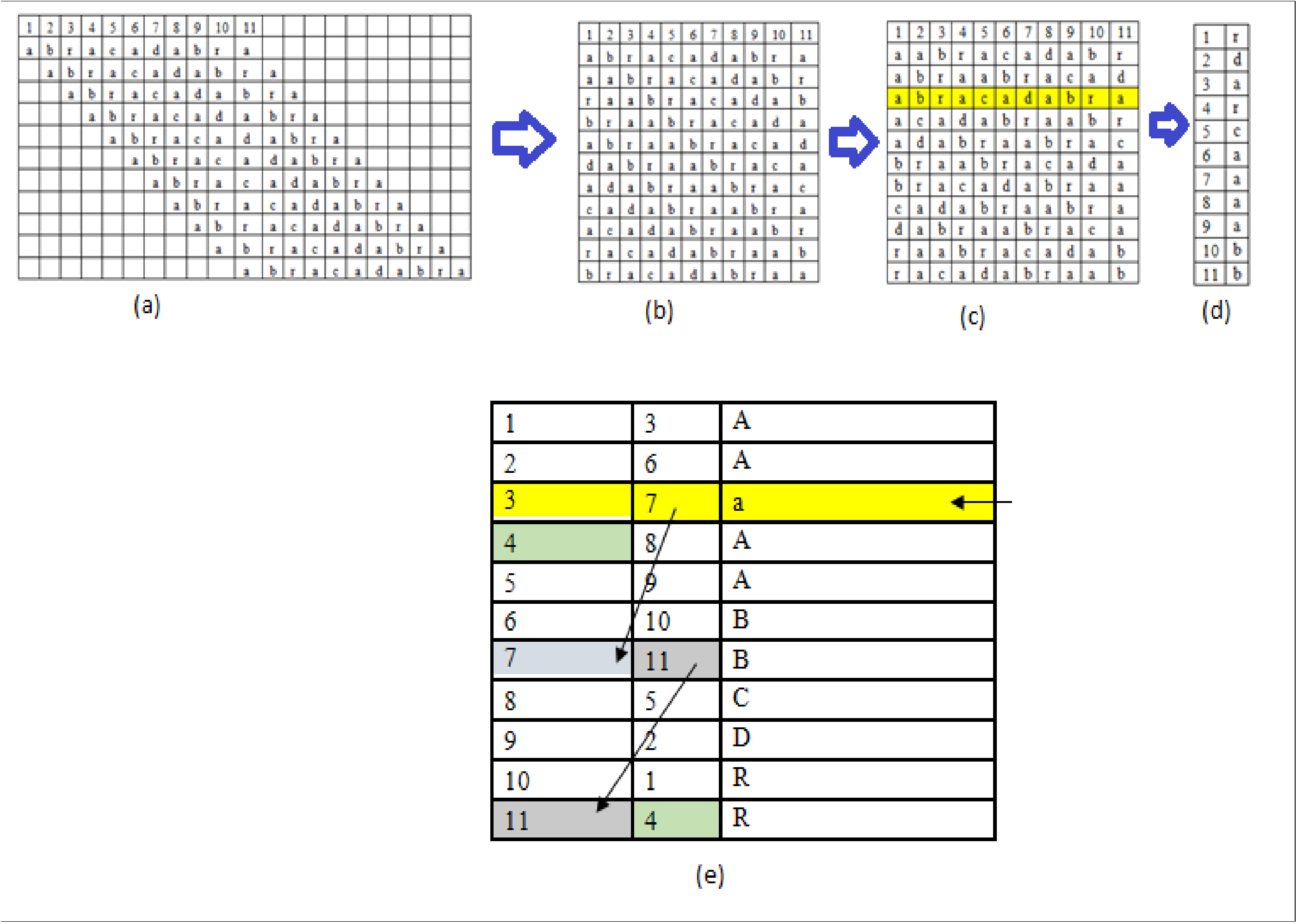


Figure 16: Burrows- Wheeler Transform example at work

Notice that the original word is in row #3, the encoded word is the last column. The pair will send over the medium which is (rdarcaaaabb,3). To decode the word the following is to be carried out: First, index the coded word like figure 16.d. Second, sort the matrix in alphabetical order and index it again as seen in the below form, figure 16.e:

To decode, start decoding from row #3, use the second column to point to the next letter First letter must be "a", the next letter is in row #7 as the second column suggests which is "b", the third letter is indexed in row 11, etc.

MOVE TO FRONT (MTF) TRANSFORM

MTF was first published by Ryabko and is used as an extra step in encoding algorithm to improve the performance of entropy encoding techniques of compression Ryabko, 1980 and again published in (Ryabko, 1980) . Next section presents the MTF algorithm, and a classical working example of MTF.

A) MTF Algorithm

The algorithm of MTF shown in figure 17, for both coding and decoding. The time complexity of MTF is O(n), where n is the number of symbols in the input sequence.

```
Encode
for each symbol of the input sequence:
    output the index of the symbol in the symbol table
    move that symbol to the front of the symbol table
Decode
# Using the same starting symbol table
for each index of the input sequence:
    output the symbol at that index of the symbol table
    move that symbol to the front of the symbol table
```

Figure 17. MTF Algorithm of encoding and decoding

The process of the MTF is very simple two structures are needed: one for all symbols used in in the intended to be coded input sequence, the second is the input sequence itself. The algorithm steps in the input sequence character by character. For each character the algorithm look-up the position of the character from input sequence from the symbols table. Next the algorithm moves the symbol at hand to the beginning of the symbols table. To decode, the exact opposite takes place. For each symbol in the coded input sequence, look-up the symbol of the position of the coded input sequence from the look-up symbols table, next move that symbol to the front of the symbols table.

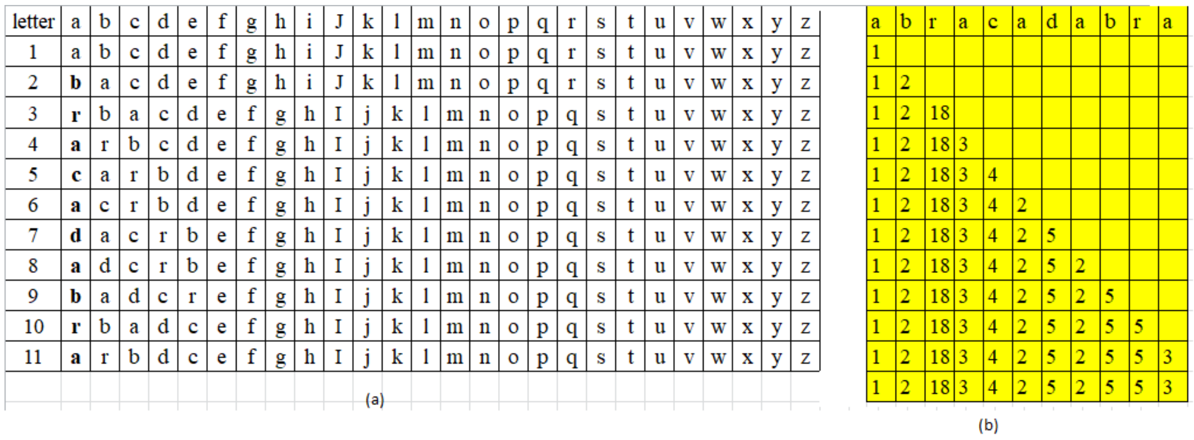
B) Classical Example of MTF

First create an alphabet queue as shown in the first row of the matrix in figure 18.a. To code the word abracadabra shown in the matrix in figure 18.b. record the position of the letter from the coded word and move that letter to the front of the alphabetical queue, as shown in figure 18.b second line. The first letter is "a" record its position "1" ass seen below, and move "a" to front of the queue. The queue stays the same since "a" is the first letter in the alphabets. Next letter is "b", record its position in the alphabets and move "b" to the front of the queue, as seen third line in matrix figure 18.b. The queue now is as in figure 18.b third line. The third letter is "r", which is in position 18, record the position and move the r to the front of the queue. See the fourth line in both matrices in figure 18.a and b. The fourth letter is "a" the position is 3, hence the letter “a” moves to beginning of the letters que and 3 is add to matrix 18.b.

The fifth letter is "c" the position is 4, and the index of "c: is 4 and both matrices are updated accordingly in the 5th line. The sixth letter is "a" the position is 2, hence both matrices are updated accordingly. The letter "a" is moved to the beginning of the letter que and its position is registered in matrix b. The seventh letter is "d" and the position is 5, both matrices are updated where the letter "d" is moved to front of the letters que and the position "5" is registered in matrix b, see figure 18 line 8.

The eighth letter is "a" and the position its is 2 in the letter que, hence its position is recorded in matrix b and the letter itself is moved to the beginning of the que. The ninth letter is "b" and the position is 5 in the que, furthermore, position is registered in the figure 18.b matrix and the letter itself is moved to head of the Que. The ninth letter is "r" and the position is 5, the position is recorded and the queue is updated as shown in line 11 both matrices in figure 18. The tenth letter is "a" and the position is 3 the position is recorded and the queue is updated as shown in line 12 both matrices in figure 18. The coded word is in the last line in figure 18.b and is send to the corresponding person as is. Next the coded word will be decoded in the following paragraphs.

To decode the word, the process starts with original letter queue and the coded word, as shown in figure 19 first line. Look at the first item in the coded text, value =1, hence the first letter in the coded alphabet is "a", move the "a" from the alphabet. The new alphabet looks like the one below. The second in the coded text is 2 which is "b" now move "b" to beginning of the alphabet queue as shown in line 2 of figure 19.



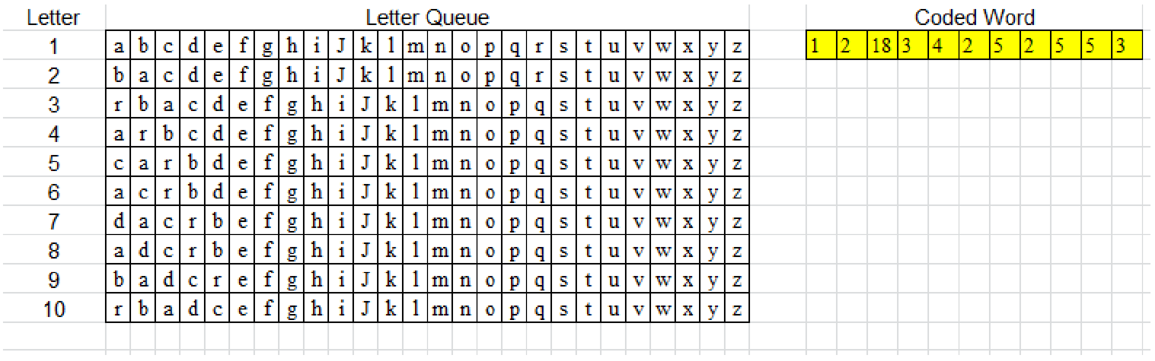
letter	a	b	c	d	e	f	g	h	i	J	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
1	a	b	c	d	e	f	g	h	i	J	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
2	**b**	a	c	d	e	f	g	h	i	J	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
3	**r**	b	a	c	d	e	f	g	h	I	j	k	l	m	n	o	p	q	s	t	u	v	w	x	y	z
4	**a**	r	b	c	d	e	f	g	h	I	j	k	l	m	n	o	p	q	s	t	u	v	w	x	y	z
5	**c**	a	r	b	d	e	f	g	h	I	j	k	l	m	n	o	p	q	s	t	u	v	w	x	y	z
6	**a**	c	r	b	d	e	f	g	h	I	j	k	l	m	n	o	p	q	s	t	u	v	w	x	y	z
7	**d**	a	c	r	b	e	f	g	h	I	j	k	l	m	n	o	p	q	s	t	u	v	w	x	y	z
8	**a**	d	c	r	b	e	f	g	h	I	j	k	l	m	n	o	p	q	s	t	u	v	w	x	y	z
9	**b**	a	d	c	r	e	f	g	h	I	j	k	l	m	n	o	p	q	s	t	u	v	w	x	y	z
10	**r**	b	a	d	c	e	f	g	h	I	j	k	l	m	n	o	p	q	s	t	u	v	w	x	y	z
11	**a**	r	b	d	c	e	f	g	h	I	j	k	l	m	n	o	p	q	s	t	u	v	w	x	y	z

(a)

a	b	r	a	c	a	d	a	b	r	a
1										
1	2									
1	2	18								
1	2	18	3							
1	2	18	3	4						
1	2	18	3	4	2					
1	2	18	3	4	2	5				
1	2	18	3	4	2	5	2			
1	2	18	3	4	2	5	2	5		
1	2	18	3	4	2	5	2	5	5	
1	2	18	3	4	2	5	2	5	5	3
1	2	18	3	4	2	5	2	5	5	3

(b)

Figure 18. A classical example of MTF.



Letter	Letter Queue																									
1	a	b	c	d	e	f	g	h	i	J	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
2	b	a	c	d	e	f	g	h	i	J	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
3	r	b	a	c	d	e	f	g	h	i	J	k	l	m	n	o	p	q	s	t	u	v	w	x	y	z
4	a	r	b	c	d	e	f	g	h	i	J	k	l	m	n	o	p	q	s	t	u	v	w	x	y	z
5	c	a	r	b	d	e	f	g	h	i	J	k	l	m	n	o	p	q	s	t	u	v	w	x	y	z
6	a	c	r	b	d	e	f	g	h	i	J	k	l	m	n	o	p	q	s	t	u	v	w	x	y	z
7	d	a	c	r	b	e	f	g	h	i	J	k	l	m	n	o	p	q	s	t	u	v	w	x	y	z
8	a	d	c	r	b	e	f	g	h	i	J	k	l	m	n	o	p	q	s	t	u	v	w	x	y	z
9	b	a	d	c	r	e	f	g	h	i	J	k	l	m	n	o	p	q	s	t	u	v	w	x	y	z
10	r	b	a	d	c	e	f	g	h	i	J	k	l	m	n	o	p	q	s	t	u	v	w	x	y	z

Coded Word										
1	2	18	3	4	2	5	2	5	5	3

Figure 19. Decoding using MTF.

The third in the coded text is 18 which is "r" in the alphabet, again move it to the front of the alphabet, like line 3, in figure 19. The fourth is number 3 which is "a" in alphabets, again move the "a" to beginning of the alphabets queue line 4 figure 19. The fifth in the coded word is 4 which is "c" in the alphabets queue. Move the c to the front of the alphabets queue as shown in line 5. The sixth coded letter is 2 which "a" in the alphabets above, again move a to front of the alphabets queue shown in line 6. The seventh is 5, which "d" in the alphabets queue, and move the d to the front of alphabets queue as in line 7 in figure 19. The eighth is 2 which is "a" in the alphabets, and move the "a" to the front of letter queue, seen line 8. The ninth is 5 in the coded word which is "b" in the alphabets, again move "b" to the front of the alphabets queue. The tenth is "5" which is "r" in the alphabets above, and move "r" to front of the alphabets queue. The eleventh is 3, which is a in the alphabet above, and move the "a" to the front. Hence producing the decoded word: **abracadabra**.

HAAR WAVELET TRANSFORM

Haar was proposed in 1910 by the Hungarian mathematician Alfréd Haar; hence it is one of the oldest transform functions. The basic philosophy of the Haar is to create two functions using a 2X2 matrix: one function adds two variables and calculates the average; the second function calculates the difference between the variables and calculate the difference average. In the next sections: first the steps of forward transform & revers transform are explained, then an example of the forward transform is given with real numbers. The revers transform is explained in details with an example.

A) The Haar Process

In this section, the Haar process is explained. There are two major steps conducted in the process: Forward transform and revers transform. The forward transform is represented in steps 1 & 2, and then the revers transform is explained in steps 3 & 4, as shown below

The following will be conducted:

1. *Create D matrix from C, where C is a 2X2 matrix*

D-Col#1=(C-Col#1+C-Col#2)/2,
D-Col#2= (C-Col #3+C-Col#4)/2

D-Col#3= (C-Col #1-C-Col #2)/2,
D-Col#4= (C-Col #3-C-Col #4)/2

2. *Use the D matrix to create the new F matrix as shown below:*

F-Row#1= (D-Row 1+D-Row 2)/2,
F-Row#2= (D-Row 3+ D-Row 4)/2

F-Row#3 = (D-Row 1-D-Row 2)/2,
F-Row#4 = (D-Row 3- D-row 4)/2

The resulting matrix F is the coded matric created from the original matrix C. To reverse the whole process and to prove that we can reproduce the original matrix C, the following steps are to be followed:

3. Revers operation create R matrix by using matrix F from the previous step, as follows:

R-Row#1= F-Row1+Row 3,
R-Row#2= F-Row 1-F-Row 3

R-Row#3= F-Row 2+ F-Row 4,
R-Row#4= F-Row 2- F-Row 4

4. Create E Matrix by:

E-Col#1=R-Col#1+R-Col#3,
E-Col#2=R-Col 1-R-Col 3

E-Col#3= R-Col 2+R-Col 4;
E-Col#4= R-Col 2+R-Col 4

B) Example Forward Transfer

In this section, the forward transform is shown in an example using real numbers to show the process clearly. Say we have the following 4X4 matrix named C. Each element in C will be as shown in the matrix C below

C=

Col 1	**Col 2**	**Col 3**	**Col 4**
100	50	60	150
20	60	40	30
50	90	70	82
74	66	90	58

First, calculate the average & distance for each row, as follows:

Designate the first two columns for the average and the two columns for the distance for every two consecutive elements in a row. Hence

D (1,1) = (C (1,1) +C (1,2))/2,
D (1,2) = (C (1,3) +C (1,4))/2.

For the distance (the second two columns) we do the following

D (1,3) = (C (1,1)-C (1,2))/2,
D (1,4) = (C (1,3)-C (1,4))/2.

For the second row, again we do the following:

D (2,1) = (C (2,1) +C (2,2))/2,
D (2,2) = (C (2,3) +C (2,4))/2.

For the distance (the second two columns) we do the following:

D (2,3) = (C (2,1)-C (2,2))/2,
D (2,4) = (C (2,3)-C (2,4))/2.

For the third row, we do again the following:

D (3,1) = (C (3,1) +C (3,2))/2,
D (3,2) = (C (3,3) +C (3,4))/2.

For the distance (the second two columns) we do the following:

D (3,**3**) = (C (3,1)-C (3,2))/2,
D (3,**4**) = (C (3,3)-C (3,4))/2.

Last for row number 4 we do the following:

D (4,1) = (C (4,1) +C (4,2))/2,
D (4,2) = (C (4,3) +C (4,4))/2.

For the distance (the second two columns) we do the following:

D (4,**3**) = (C (4,1)-C (4,2))/2,
D (4,**4**) = (C (4,3)-C (4,4))/2.

The result with addition and subtraction is shown in the matrix D, below:

D=

(Col 1+Col 2)/2	**(Col 3+Col 4)/2**	**(Col 1-Col 2)/2**	**(Col 3-Col 4)/2**
(100+50)/2=75	(60+150)/2=105	(100-50)/2=25	(60-150)/2=-45
(20+60)/2=40	(40+30)/2=35	(20-60)/2=-20	(40-30)/2=5
(50+90)/2=70	(70+82)/2=76	(50-90)/2=-20	(70-82)/2=-6
(74+66)/2=70	(90+58)/2=74	(74-66)/2=4	(90-58)/2=16

Hence, the result is simply shown below:

D=

Row1	75	105	25	-45
Row 2	40	35	-20	5
Row 3	70	76	-20	-6
Row 4	70	74	4	16

Second step is, we work with the D matrix to create the new F matrix shown below: We store the average in the first two rows of F matrix and the distance in the lower two rows: Hence,

F (1,1) = (D (1,1) +D (2,1))/2,
F (1,2) = (D (1,2) +D (2,2))/2,

F (1,3) = (D (1,3) +D (2,3))/2,
F (1,4) = (D (1,4) +D (2,3))/2.

The second row in F is calculated as follows:

F (2,1) = (D (3,1) +D (4,1))/2,
F (2,2) = (D (3,2) +D (4,2))/2,

F (2,3) = (D (3,3) +D (4,3))/2,
F (2,4) = (D (3,4) +D (4,3))/2.

The third row in F, is designated for distance and is calculated as follows:

F (3,1) = (D (1,1) -D (2,1))/2,
F (3,2) = (D (1,2) -D (2,2))/2,

F (3,3) = (D (1,3) -D (2,3))/2,
F (3,4) = (D (1,4) -D (2,3))/2.

The fourth row in F, is designated for distance and is calculated as follows:

F (4,1) = (D (3,1) -D (4,1))/2,
F (4,2) = (D (3,2) -D (4,2))/2,

F (4,3) = (D (3,3) -D (4,3))/2,
F (4,4) = (D (3,4) -D (4,3))/2.

(Row1+row2)/2	(75+40)/2=57.5	(105+35)/2=70	(25+-20)/2=2.5	(-45+5)/2=-20
(Row 3+ row 4)/2	(70+70)/2=70	(76+74)/2=75	(-20+4)/2=-8	(-6+16)/2=5
(Row1-row2)/2	(75-40)/2=17.5	(105-35)/2=35	(25--20)/2=22.5	(-45-5)/2=--25
(Row 3- row 4)/2	(70-70)/2=0	(76-74)/2=1	(-20-4)/2=-12	(-6-16)/2=-11

The end result is the following matrix shown below name F. matrix F when the reverse function is used will return to the original matrix.

F=

Row 1	57.5	70	2.5	-20
Row 2	70	75	-8	5
Row 3	17.5	35	22.5	-25
Row 4	0	1	-12	-11

C) The Revers Transfer of Haar

The revers process is conducted on the matrix F produced previously. The result of the revers is show in the matrix R below and will be explained next. First, each element in first row of matrix F is added to the third row (row3). We name the result matrix from the operation the R matrix for reference purposes. Hence:

R (1,1) =F (1,1) +F (3,1),
R (1,2) =F (1,2) +F (3,2),

R (1,3) =F (1,3) +F (3,3),
R (1,4) =(F1,4) +F (3,4).

Second, each element in first row of F matrix will be subtracted from the third row of F matrix, as shown below

R (2,1) =F (1,1)-F (3,1),
R (2,2) =F (1,2)-F (3,2),

R (2,3) =F (1,3)-F (3,3),
R (2,4) =(F1,4)-F (3,4).

The third row of R is adding the element of second and fourth row of F matrix, as shown below:

R (3,1) =F (2,1) +F (4,1),
R (3,2) =F (2,2) +F (4,2),

R (3,3) =F (2,3) +F (4,3),
R (3,4) =(F2,4) +F (4,4).

The fourth row of R is subtracting the element of second row and fourth row of matrix F, as shown below:

R (4,1) =F (2,1)-F (4,1),
R (4,2) =F (2,2)-F (4,2),

R (4,3) =F (2,3)-F (4,3),
R (4,4) =(F2,4)-F (4,4).

All addition and subtract are included in the matrix R below:

R=

Row1+Row 3	57.5+17.5=75	70+35=105	2.5+22.5=25	-20+-25=-45
Row 1-row 3	57.5-17.5=40	70-35=35	2.5-22.5=-20	-20—25=5
Row 2+ Row4	70+0=70	75+1=76	-8+-12=-20	5+-11=-6
Row 2- row 4	70-0=70	75-1=74	-8- -12=4	5- -1=16

The result is matrix R which one step short for complete reversal:

R=

Col 1	Col 2	Col3	Col 4
75	105	25	-45
40	35	-20	5
70	76	-20	-6
70	74	4	16

The final step is column operations which is conducted on the R matrix, the first column and the third column are add, then first column and third column are subtracted. Furthermore, the same for second column and fourth column. Hence the rule: Add every other column and subtract every other column. All the steps will be explained in the following:

The end result matrix we will call E matrix

E=

Col 1+Col 3	Col 1-Col 3	Col 2+Col 4	Col 2-Col 4
75+25=100	75-25=50	105+-45=60	105- -45=150
40+-20=20	40- -20=60	35+5=40	35-5=30
70+-20= 50	70- -20=90	76+-6=70	76 - -6 =82
70+4= 74	70-4=66	74+16=90	74-16 = 58

First step is to add every element in first column and second column respectively, as follows:

E (1,1) =R (1,1) +R (1,3), E (2,1) =R (2,1) +R (2,3)

E (3,1) =R (3,1) +R (3,3), E (4,1) =R (4,1) +R (4,3)

The second column is the subtract operation which is conducted as follows:

E (1,2) =R (1,1)-R (1,3), E (2,2) =R (2,1)-R (2,3)

E (3,2) =R (3,1)-R (3,3), E (4,2) =R (4,1)-R (4,3)

Third step is to add Col2 and column 4 as follows:

E (1,3) =R (1,2) +R (1,4), E (2,3) =R (2,2) +R (2,4)

E (3,3) =R (3,2) +R (3,4), E (4,3) =R (4,2) +R (4,4)

The fourth column is the subtract operation which is conducted as follows:

E (1,4) =R (1,2)-R (1,4), E (2,4) =R (2,2)-R (2,4)

E (3,4) =R (3,2)-R (3,4), E (4,4) =R (4,2)-R (4,4)

WAVELET TREE

Wavelet Tree, a succinct data structure for storing a compressed sequence developed by R. Grossi and A. Gupta in 2003 (Gross, Gupta, & Vitter, 2003). In the next section the algorithm is shown along with and example that shows position search, symbol search and Bottom-to-top search.

A) Build the Wavelet Tree Algorithm

The algorithm of the building a wavelet tree is recursive simple algorithm as shown in figure 20. The example following the algorithm will show how intuitive the algorithm is. The following is the algorithm put simply:

Scan the alphabet of the text and encode the first half to zero and the second half to one.

Group each 0-encoded symbol as sub tree.

Group each 1-encoded symbol as sub tree.

Reapply this to each sub tree recursively until there are only leaves.

Figure 20. Build the Wavelet Tree algorithm.

B) Classical Example Wavelet Tree

Suppose we want to code the famous word "abracadabra", first scan the word for the alphabets used in "abracadabra", we find that the 5 letters are "abrcd". Next, split the letters "abrcd" into two groups: "ab" and "rcd" code the first group to zero (0) and the second group to one (1). Replace, the letters with the respective code, as follows

a	b	r	a	c	a	d	a	b	r	a
0	0	1	0	1	0	1	0	0	1	0

Now we build the original tree, figure 21. At the root of the tree is the previously code word as seen in the graph below: split the word into two parts: part the "ab" letters as they are shown in the root, and part with all "cdr" as shown in the root. This operation will produce two subtrees: "abaaaba" coded "0100010"and the subtree "rcdr " coded "0110". The first sub tree will produce two leaves the "a" & "b" leaves. The second subtree "rcdr" will be coded "0110" will produce the leaf "r" and sub tree "cd" again code "01".

To make sure that our coding is right; we will ask the following question, what is the letter in the 7th position? To answer the question, we look at the 7th position value, it is "1". We at the tree above at the branch labeled "1" and also count how many 1s before the 7th position. There are two (2) ones (1s) before the 7th position. Now, follow the branch labeled "1" from the root. The sub tree "rcdr" is located at the end of that branch. Examine the subtree and count, the second one in the subtree, and follow the branch labeled "1" while remembering how many ones (1s) are before that digit "1". The branch will lead to sub tree "cd", again examine the sub tree, find the first one (1), and follow the branch labeled 1. The end leaf is "d" which is the answer we are look for.

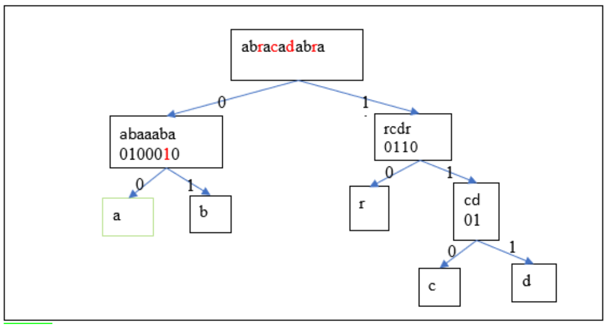


Figure 21. The tree produced from the Wavelet Tree algorithm.

Another example, find the letter in the 10th position? First find the value of the 10th position, which is "1" and there are 3 ones"1s" before it. Next, move down the branch labeled 1 and in the coded tree find the 4th digit, the 4th digit is "0". Follow the branch labeled "0", at the end of the branch a leaf of letter "r" is found. Hence, the letter in the 10th position in text is the letter "r".

C) Bottom-To-Top Search

Find the second letter "b" in the coded word using the tree. Start with the leaf labeled "b", climb up the branch, the branch is labeled "1", now look for the **2ed** "1" in the sub tree. The position of the second "1" in the subtree is **6th** position. Again, climb once more through the branch labeled "0"and remember the number 6. Once you reach, the higher subtree count to the **6th zero** in the code of the higher sub tree.

Another example, find the 5th "a" in the code. First, we start with leaf "a", and climb up the branch "0", we find the 5th zero it is in the 7th position. We climb again through the branch which labeled zero. Now we look for the 7th zero in the code which the last bit and it is the code for the 5th "a".

DELTA ENCODING

Delta encoding is also known as: Delta compression and Data Differencing. Simple delta encoding is transmitting or storing data in the form of differences (deltas) between sequential data rather than the value of data itself. Storing and transmitting the differences lightens the load and reduces the amount of data variance. The algorithm, figure 22, is also a very simple intuitive algorithm. To store or transmit two values 99, 100 the transmitted data will be 99, 1, hence transmitting 3 digits rather than 5 digits. A simple example of using Delta encoding is consider the following sequence 2 4 6 8 9 10 7 will stored as follows: 2 2 2 2 1 1 -3 basically the value stored is the previous value subtracted from the consequent value hence the name Delta.

```
For i=2 to end of sequence
        Sequence[i] = Sequence[i-1]-Sequence[i]
End loop
```

Figure 22. Delta algorithm.

RICE & GOLOMB CODING

Rice coding was developed by Robert F. Rice in 1979 and is (Golomb, 1966) a subset from Golomb coding which developed by Solomon W. Golomb in 1966 as an alternative to Huffman coding (Golomb, 1966). Rice coding is an important yet simple algorithm. Rice coding is used in: shorten, FLAC, Apple Lossless, MPEG-4 ALS, JPEG-LS, and FELICS. Hence it is imperative to discuss Rice coding. Rice coding is a subset of Golomb codes, when M is power 2 type.

A) The Algorithm

The algorithm below is a simplest version of the Rice & Golomb algorithm (Rice, 1979). The algorithm, figure 23, is best seen through the example as follows.

B) The Rice Coding in Simple Steps.

The following is an example that shows the algorithm live work. The simple operation of breaking a number into quotient and remainder. Suppose, N is presented, where N is the message needs to be coded given an M. M is a number usually given for such a problem.

Suppose that M=10.

N: the message needs to be codded.

First, Divide N to quotient (Q) and remainder (R). The Q is represented by unary coding followed by zero. The R is represented by binary.

$$Q = \left\lfloor \frac{N}{M} \right\rfloor$$

R= N mod M

Second, calculate $b = \lceil \log_2 M \rceil$ then check the following:

```
1.  Fix the parameter $M$ to an integer value.
2.  For $N$, the number to be encoded, find
quotient = $q$ = int[$N/M$]
remainder = $r$ = $N$ modulo $M$
3.  Generate Codeword
1.  The Code format: <Quotient Code><Remainder Code>, where
2.  Quotient Code (in unary coding)
1.  Write a $q$-length string of 1 bits
2.  Write a 0 bit
3.  Remainder Code (in truncated binary encoding)
1.  If $M$ is power of 2, code remainder as binary format. So $\log_2(M)$ bits are needed. (Rice code)
2.  If $M$ is not a power of 2, set $b = \lceil \log_2 M \rceil$
1.  If R < ($2^b$-M) then code R in (b-1) binary Bits
2.  If R>= ($2^b$-M) then code the number R+$2^b$-M in binary representation using b bits.
```

Figure 23. Golomb & Rice algorithm (Rice, 1979).

- If M is of power 2, code R in binary format, so $\log_2 M$ bits are needed (Rice)
- If R< 2^b-M Code R in b-1 binary bits.
- If R >= 2^b-M code the number R+2^b-M in binary representation using b bits.

To calculate $b = \lceil \log_2 10 \rceil$=4
If R < (2^4-10 =6) then code R in (b-1) which is (3) binary Bits

If R>= (2^4-10 =6) then code the number R+2^4-10 in binary representation using b bits.

QUOTIENT			**REMAINDER**				
Q	**Output Bits**		**R**	**Offset**	**Binary**	**Output bit Binary**	**Note**
0	0		0	0	0000	000	R<6
1	10		1	1	0001	001	Rule 1
2	110		2	2	0010	010	Use 3 bits
3	1110		3	3	0011	011	
4	11110		4	4	0100	100	
5	111110		5	5	0101	101	
6	1111110		6	6+6=12	1100	1100	R>=6
			7	7+6=13	1101	1101	Rule 2
N	111..110		8	8+6=14	1110	1110	Use 4 bits & Code R+6
			9	9+6=15	1111	1111	

For example, if the coded word is 42 then the quotient is 4 and the R is 2, hence the coded word is <11110>;<010>. Notice that in the remainder less than 6 the first rule is invoked and the R is coded using b-1 bits or if the word is 67, then Q=6 and R=7, hence the coded word is <1111110>;<1101>. Notice that since the remainder greater than 6 the second rule is invoked and the coded R is the R+2^4-10 which is the offset =13 in other words 7+16-10=13 offset.

To *decode* a coded word, say "11111101111" given M=10 we do the following?

- First, we can guess that the Q= 6 since the word started with 6 ones.
- Second, we can calculate b from the given M, which is b=4.
- To reverse the rules 1 and 2 since the R is 4 bits' length then the code must have used the second rule. Hence the R+6=15 which is 9. Hence the coded word is 69.

TUNSTALL CODING

Tunstall Coding was developed by Brian Parker Tunstall as part of his PhD thesis in 1967. Tunstall coding builds a tree for all possible combinations of the symbols used in a text. The tree is built according the frequency of the symbols hence the symbol with highest probability is to branch out in the tree. In the next two sections: the algorithm, figure 24 is explained and an example that shows the work of the algorithm. The variable *u*: is an input string. The *D* is the constructed dictionary as tree probabilities; each branch is associated with a letter from input alphabets.

The *C* is upper bound to the size of the dictionary.

```
D := tree of |u| leaves, one for each letter in u.
 While |D|<C:
  Convert most probable leaf to tree with |u| leaves.
```

Figure 24. Tunstall Coding algorithm.

For example, to code the following text "Hello world." using Tunstall algorithm the following is to be done: First, scan the text and find the alphabet and calculate the frequency of each character, as shown below in table 4:

Table 4. Frequency of Tunstall's scan.

Letter	**Frequency in text**	**Portability**
H	1	1/12
E	1	1/12
L	3	3/12
0	2	2/12
W	1	1/12
R	1	1/12
d	1	1/12
Sp	1	1/12
period	1	1/12

Second, since there are |u| = 9 then each symbol can be represented with $\log_2 9$ = 4 bits, and build a tree with one root and 9 branches as figure 25:

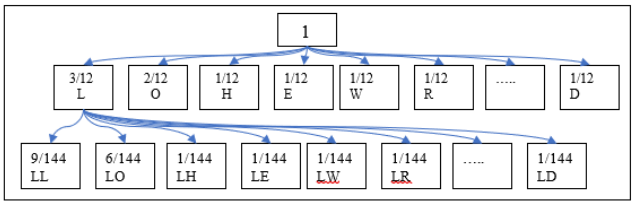


Figure 25. Tree of Tunstall Coding.

Next, we branch for the leave L since it has the highest probability. We don't stop branching since the number of leave 16 >28-9. But the dictionary is as follows:

H	00000
E	00001
LL	00010
LE	00011
LH	00100
LO	00101
LW	00110
LR	00111
LD	01000
L SP	01001
L PERIOD	01010
0	01011
W	01100
R	01101
d	01110
Sp	01111
period	10000

Looking back at the text "hello world" of length 12 symbols, each symbol must have been coded with 8 bits. The total coded should have been 12*8=96 bits. Using Tunstall coding each symbol was coded with 5bits and the symbol "LL" in the word "hello" was also code by 5 bits, hence the total number of bits is 5*8+5=45 bits. The compression ratio is 1-45/96=1-0.468=53%.

HYBRID TYPES

Three hybrid compression algorithms are discussed in this section: bzip2, deflate, Run-Length Golomb-Rice (RLGR)

A) bzip2

Hybrid types are quite common in the compression world. Hybrid types are the compression algorithms that use more than one method and technique and algorithms to come up with software that compresses using the advantage of each algorithm. An example of hybrid types is bzip2. bzip2 is a mix of compression algorithms and methods, bzip2 uses Run-Length Encoder, Burrows-Wheeler Transform, Move-To-Front Transform, and Huffman code. The steps carried out as follows according to Seward (Seward, 2000):

bzip2 algorithm

The algorithm or mix of algorithms named bzip2, figure (26):

DEFLATE

DEFLATE invented in 1993 by Phil Katz. Deflate is basically LZ&& and LZSS and Huffman Code. Hence, DEFLATE is a combination of three compression methods.

Run-Length Golomb-Rice (RLGR)

Run-Length Golomb-Rice (ARLGR) developed and used in Microsoft research center was a natural development to Rice & Golomb coding which is really adding the algorithms together. Hence, ARLGR is considered hybrid algorithm. The ARLGR was developed by Malvar in 2006 (Malvar, 2006).

GENEALOGY TREE

The following tree, figure 27, tells the story of 40 compression algorithms. Starting with Haar Wavelete transform (1910), Arithmetic coding of Shannon developed in 1948, then Fano in 1949. After Fano's Peter's version was developed and *Run Length Encoding* in 1967. In 1973 published another version named Enumerative Coding. In 1976 (Rissanen, 1976) introduced Last in First Out (LIFO) version of the algorithm. In 1976 Pasco (Pasco, 1976) introduced the FIFO version of the algorithm. In 1979 "stream" code was discovered by Rubin (Rubin, 1979) as an improvement of Pasco's work. Martin (1979) and Jones (1981) developed P-based FIFO arithmetic codes (Jones, 1981) and (Martin, 1979). In 1977 LZ was first developed in 1977 by Jacob Ziv and Abraham Lempel, and was name LZ77. In 1978, both Ziv and Lemple, developed the algorithm and named it LZ78. Many LZ based algorithms were developed over the years: in 1981 LZR was developed, in 1982 LZSS, in1984 LZW, in 1985 LZJ, LZC, and LZZMW; in 1987 LZB, LZH, LZT. In 1988 LZAP was developed, in 1991 LZRW, in 1993 DEFLATE, in 1994 LZS, in 1995 LZP, & LZX, in 1996 LZO, in 1998 LZMA & LZJB, in 2006 LZWL in 2009 LZMA2 and in 2011 LZ4. Move to Front (MTF) was developed in 1980. MTF was first published by Ryabko and is used as an extra step in encoding algorithm to improve the performance of entropy encoding techniques of compression. Chronologically LZS and Burrows-Wheeler Transform were developed. Also, table 5 recaps the results of researching the 12 coding algorithms.

CONCLUSION

This paper presented the pillars of compression algorithms, methods and techniques. The paper counted more than 40 compression algorithm: Arithmetic Coding - SHANNON (1948), Huffman coding (1952), FANO (1949), Run Length Encoding (1967), Peter's Version (1963), Enumerative Coding (1973), LIFO(1976), FiFO Pasco(1976), LZ77(1977), Move To Front (MTF)Transform (1980), LZ78(1978), LZR (1981), LZSS (1982), LZJ (1985), LZC(1985), LZZMW(1985), LZB (1987), LZH (1987), LZT (1987), LZAP (1988), LZRW (1991)), DEFLATE(1993), LZS(1994), LZP(1995), LZX (1995), LZO (1996), LZMA, LZJB, LZWL(2006), LZMA2(2009),

First, a Run-Length Encoder is applied to the data.

Next, the Burrows-Wheeler Transform is applied.

Then, a Move-To-Front Transform is applied with the intent of creating a large number of identical symbols forming runs for use in yet another Run-Length Encoder.

Finally, the result is Huffman coded and wrapped with a header

Figure 26. bzip2 Algorithm (Seward, 2000)**.**

Table 5. The summery table of 12 coding algorithms, where n is length of text**.**

Coding Algorithm	Year	Time complexity	Developer	Philosophy	
SHANNON–FANO CODE	1948 1949	O (n + |symbols| * log| symbols |)	Shannon Fano	Recursive	Frequency & Tree
Arithmetic Coding		O(|symbols|+n)		Probability Split intervals	Frequency
HUFFMAN CODE	1952	O (|symbols| log |symbols|)	David A. Huffman	Priority list & frequency	Tree
LZ	1977	O(n)	Jacob Ziv and Abraham Lempel	Dictionary	
LZSS	1982		James Storer and Thomas Szymanski	Dictionary with offset	
RUN LENGTH ENCODING (RLE)	1967	O(n)		Count of each repeated symbol	
BURROWS- WHEELER TRANSFORM	1994	O (n log n)	BURROWS-WHEELER	Matrix warp	
MOVE TO FRONT (MTF)TRANSFORM	1980	O(n)	Ryabko	Queue of symbols & their order	
Haar Wavelete transform	1910	O(n)	Haar	2 functions & 2 unknowns	Matrix 2X2
WAVELET TREE	2003	O (n log symbols)	R. Grossi and A. Gupta	Binary tree	
DELTA ENCODING		O(n)		Difference between two symbols	
RICE & GOLOMB CODING	1967 1979		Golomb & Rice	Mod & remainder	
TUNSTALL CODING	1967	O (symbols log symbols)	Tunstall	Tree & Maximum probability of symbol	

LZ4(2011), Burrows- Wheeler Transform (1994), Haar (1910), Wavelet tree (2003), Stream (1979), P-Based FIFO (1981), Delta Encoding, Rice & Golomb Coding (1966,1979), Run-Length Golomb-Rice (RLGR) (2007), Tunstall coding (1967). Although each algorithm is an independent in its own right, still; these algorithms interrelate genealogically and chronologically. Hence, the major stubs in the developed tree of the compression algorithms are 12. The tree is presented in the last section of the paper after presenting the 12 main compression algorithms each with a practical example.

The paper first introduced Shannon–Fano code showing its relation to Shannon (1948), Huffman coding (1952), Fano (1949), Run Length Encoding (1967), Peter's Version (1963), Enumerative Coding (1973), LIFO (1976), FiFO Pasco (1976), Stream (1979), P-Based FIFO (1981). Two examples are presented one for Shannon-Fano Code and the other is for Arithmetic Coding. Next, Huffman code was presented with simulation example and algorithm. The third is Lempel-Ziv-Welch (LZW) Algorithm which hatched more than 24 algorithms. The LZW family was presented and a working example was given. The fourth is Run Length Encoding (RLE) which essentially codes using frequency and position. The fifth is Burrows-Wheeler Transform. The sixth is Move-to-Front (MTF) Transform. The seventh is Haar the eighth is wavelet tree. The ninth is the Delta Encoding, and the tenth is Rice & Golomb Coding. The eleventh is Tunstall coding. The twelfth is a hybrid example bzip2 which is a mix of many algorithms & DEFLATE algorithm which is also a mix of three algorithms. The last example of the hybrid is Run-Length Golomb-Rice (RLGR).

The paper then presented the genealogy tree suggested by the paper. The tree shows the interrelationships between the 40 algorithms. Also, the tree showed the chronological order the algorithms came to life. The time relation shows the cooperation among the scientific society and how they amended each other's work. Furthermore, novice users of the algorithms and researchers can find a reference where they can use this research as spring board for future work. The researcher, wanted to accompany each algorithm with a classical example to make understanding easier and recalling the algorithm much faster with an example rather than an algorithm.

REFERENCES

Abramson, N. (1963). *Information Theory and Coding.* New York: McGraw-Hill Book Co., Inc.

Burrows, M., & Wheeler, D. J. (1994). *A block sorting lossless data compression algorithm.* Digital Systems Research Center. Palo Alto, California: Digital Equipment Corporation. Retrieved from http://www.hpl.hp.com/techreports/Compaq-DEC/SRC-RR-124.pdf

Cover, T. M. (1973). Enumerative Source Coding. *IEEE Transactions on Information Theory, 19*(1), 73 - 77. doi:10.1109/TIT.1973.1054929

Cslearning. (2013, Dec 19). *Lempel-Ziv-Welch Compression Algorithm – Tutorial.* Retrieved from www.youtube.com: https://www.youtube.com/watch?v=j2HSd3HCpDs

Fano, R. (1949). *The transmission of information.* MASSACHUSETTS INSTITUTE OF TECHNOLOGY. Cambridge (Mass.), USA: Research Laboratory of Electronics at MIT. Retrieved from https://www.hcs64.com/files/fano-tr65-ocr-only.pdf

Golomb, S. (1966). Run-Length Encodings. *IEEE Transactions on Information Theory, 12*(3), 399-401. doi:10.1109/TIT.1966.1053907

Gross, R., Gupta, A., & Vitter, J. (2003). High-order entropy-compressed text indexes. *SODA '03 Proceedings of the fourteenth annual ACM-SIAM symposium on Discrete algorithms* (pp. 841–850). Baltimore, Maryland: Society for Industrial and Applied Mathematics. Retrieved from https://dl.acm.org/citation.cfm?id=644108.644250

Huffman, D. (1952). A Method for the Construction of Minimum-Redundancy Codes. *Proceedings of the IRE. 40 (30)*, pp. 1098–1101. IEEE .

Jones, C. B. (1981, May C. B. , , IT-27,280-291 (May 1981)). An Efficient Coding System for Long Source Sequences. *IEEE Trans. Info. Theory*, 280-291.

Langdon, G. (1984, March). An Introduction to Arithmetic Coding. *IBM Journal of Research and Development , 28*(2), 135-149. doi:10.1147/rd.282.0135

Lippert, R., Mobarry, C., & Walenz, B. (2005, October). A space-efficient construction of the burrows wheeler transform for genomic data. *Journal of Computational Biology, 12*(7), 943-51. doi:10.1089/cmb.2005.12.943

Malvar, H. (2006). Adaptive run-length/Golomb-Rice encoding of quantized generalized Gaussian sources with unknown statistics. *DCC '06 Proceedings of the Data Compression Conference* (pp. 23-32). Washington, DC, USA: IEEE Computer Society . doi:10.1109/DCC.2006.5

Martin, G. N. (1979). Range Encoding: an Algorithm for Removing Redundancy from a Digitized Message. *Video and Data Recording Conference.* Southampton, England,: presented at the , July . Retrieved from http://sachingarg.com/compression/entropy_coding/range_coder.pdf

Pasco, R. (1976). Source Coding Algorithms for Fast Data Compression. *Ph.D. Thesis.* CA, USA: Department of Electrical Engineering, Stanford University. Retrieved from https://www.richpasco.org/scaffdc.pdf

Rice, R. F. (1979). *Some Practical Universal Noiseless Coding Techniques.* California Institute of

Technology . Pasadena: Jet Propulsion Laboratory. Retrieved from http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.453.6684&rep=rep1&type=pdf

Rissanen, J. (1976, May). Generalized Kraft Inequality and Arithmetic Coding. *IBM Journal of Research and Development*, *20*(3), 198-203. doi:10.1147/rd.203.0198

Rubin, F. (1979, Nov). Arithmetic Stream Coding Using Fixed Precision Registers. *IEEE Transactions on Information Theory*, *25*(6), 672-675. doi:10.1109/TIT.1979.1056107

Ryabko, B. Y. (1980). Data compression by means of a "book stack”. *Problems of Information Transmission, 16*(4), 265–269. Retrieved from http://www.mathnet.ru/links/d9830cf76762a2b414a79938f7102090/ppi1458.pdf

Schalkwijk, J. (1972, May). An Algorithm for Source Coding. *IEEE Transactions on Information Theory*, *18*(3), 395 - 399. doi:10.1109/TIT.1972.1054832

Seward, J. (2000, March). *bzip2 and libbzip2*. Retrieved from bzip2 Manual: http://www.bzip.org/1.0.5/bzip2-manual-1.0.5.pdf

Shannon, C. (1948, July). A Mathematical Theory of Communication. *The Bell System Technical Journal*, *27*, 379–423. doi:10.1002/j.1538-7305.1948.tb01338.x

STORER, J. A., & SZYMANSKI, T. G. (1982, Oct). Data compression via textual substitution. *Journal of the ACM (JACM)*, *29*(4), 928-951. doi:10.1145/322344.322346

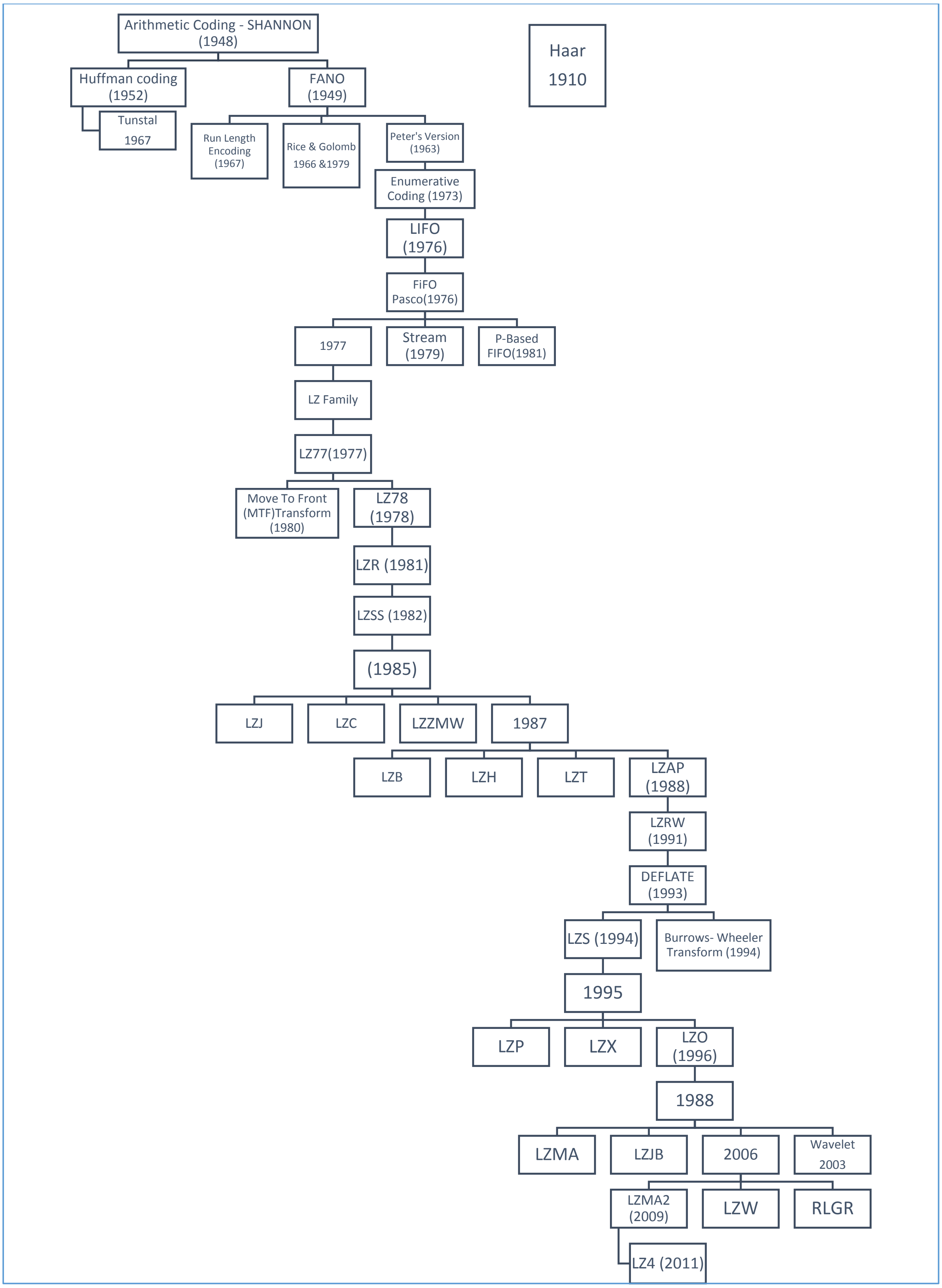


Figure 27. Genealogy Tree.