Optimizing the Performance of Text File Compression Using a Combination of the Burrows-Wheeler Transform (BWT), Move-to-Front (MTF) and Shannon-Fano Algorithms

YAYUK ANGGRAINI[1], TEDDY MANTORO[1,2], MEDIA A. AYU[2]
[1]Faculty of Science and Technology, Universitas Budi Luhur, Jakarta, Indonesia.
[2]Faculty of Engineering and Technology, Sampoerna University, Jakarta, Indonesia
[1]anggrainie@gmail.com, [2]{teddy.mantoro, media.ayu}@sampoernauniversity.ac.id

Abstract— **Compression in Information Technology is the way to minimize the file size. Performance of compression algorithm is measured by the speed of process and the compression ratio. The compression time will effect on memory allocation and CPU performance, while the low compression ratio will weakens the ability of the algorithm to compress the data. Huffman and Shannon-Fano are two compression algorithms have same ways to work, but both produced a different performance. The test results concluded that the Shannon-Fano performance has a percentage of 1,56% lower than Huffman. This problem can be solved by adding a reversible transformation algorithm to the data source. Burrows-Wheeler Transform (BWT) produces output that is more easily to be processed at a later stage, and Move-to-front (MTF) is an algorithm to transform the data unifying and reduce redundancies. This study discusses a combination of BWT + MTF + Shannon-Fano algorithm and compare it with other algorithms (Shannon-Fano, Huffman and LZ77) which were applied on text files. The test results have shown that the combination of BWT + MTF + Shannon-Fano has the most efficient compression ratio, which is 60.89% higher at around 0.37% compared to LZ77. On compression time aspect, LZ77 is the slowest, approximately 39391,11 ms, while a combination of BWT + MTF + Shannon-Fano performs at approximately 1237,95 ms. This study concluded that the combination of BWT + MTF + Shannon-Fano algorithm performs as the most optimal algorithm in compression time (speed) and compression ratio (size).**

Keyword : BWT, MTF, Shannon-Fano, Huffman, LZ77, text files, compression algorithm optimization.

I. INTRODUCTION

Information technology can be considered as a tool to create, modify, store and disseminate information. These processes produce files, where the amount of information affects the size of the file. The larger the size of the file, then the greater storage space and transmission medium are required. This can be overcome by the utilization of file compression. Data compression is the process of converting an input data stream (stream or the original raw data) into another data stream (the output, the bitstream or compressed stream) that has a smaller size [1]. Performance of compression algorithm is measured by the speed of process (compression time) and the size (compression ratio). The speed of process will give an effect on memory allocation and CPU performance, while the low compression ratio will weakens the ability of the algorithm to compress the data. In the case of the selection of unappropriate algorithm, it will lower the compression ratio and improve the execution time. Comparative study on a single algorithm, such as Huffman algorithm being the most efficient compression algorithm [2], [3], [4], [5] while the Shannon-Fano always afterwards, even though both have the similar way of compressing, but not making both produce the same performance.

One approach to achieve a better compression ratio is to develop a different compression algorithm [6], analyzing the process and the result and improve it using any possible idea. One of the alternative development approaches is by adding the transformation reversible on the data source, so that enhance the capabilities of existing algorithms to increase the compression performance. In this case, the transformation must be perfectly reversible, which means, it determines to keep the lossless nature [6] of the chosen method. Burrows-Wheeler algorithm (BW) is a lossless data compression scheme and also known as block-sorting which is one of the textual data transformation algorithm that is best in terms of speed and compression ratio until today [7]. The transformation does not process the data entries in the queue, but rather the process directly one block of text as a unit [8]. This application generated a new form and still contain the same characters so that the chance of finding the same character will be increased. The idea is to apply a transformation that is reversible to a block of text and forming a new block that contains the same characters, but are easier to be compressed with a simple compression algorithm [8], as MTF (Move-to-Front). MTF is a transformation algorithm which does not perform data compression but may help reduce redundancies, such as the result of the BWT transformation [9]. The basic idea of this method is to maintain the alphabet A of symbols as a list where frequently occurring symbols are located near the front [1]. The study in [8], [10], [7], [11], [12] were concluded that the combination between BWT with MTF was able to increase the compression ratio with increasing compression time.

This study makes the weakness on the Shannon-Fano and excess on the MTF and BWT combination as the reason for the addition of the transformation of the BWT + MTF on Shannon-Fano coding to improve the compression performance. To know the efficiency of the transformation compression process, it needs to compare the performance between the algorithms Shannon-Fano, Huffman and LZ77 algorithms combinations and BWT + MTF + Shannon-Fano.

II. RELATED WORK

A. Literature Review

1) Compression

Data compression is the science (and art) of representing information in a compact form [13]. Data compression is the process of converting an input data stream (stream or the original raw data) into another data stream (the output, the bitstream or compressed stream) that has a smaller size. A stream is either a file or a buffer in memory [1]. The data, in the context of data compression, covers all forms of digital information that can be processed by a computer program. The form of such information can be broadly classified as text, sound, pictures and video.

Any compression algorithm will not work unless a means of decompression is also provided due to the nature of data compression [13].

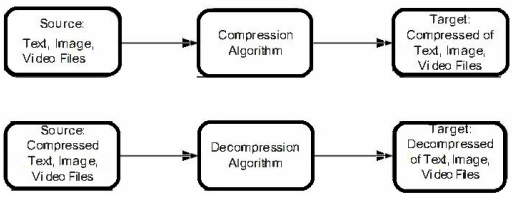


Figure 1. Compressor and decompressor.

Based on the behavior of the resulting output and outcomes, data compression techniques can be divided into two major categories, namely:

- Lossless Compression

  A compression approach is lossless only if it is possible to exactly reconstruct the original data form the compression version. There is no loss of any information during the compression process. Lossless compression is called reversible compression since the original data may be recovered perfectly by decompression [13], so the match is applied on a database file, text, medicaly image or photo satellite.

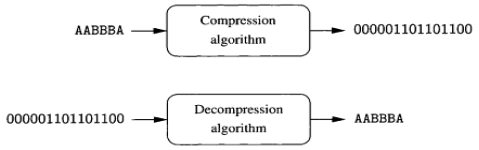


Figure 2. Lossless Compression Algorithm[13]

- Lossy Compression

  Lossy compression is called irreversible compression since it is impossible to recover the original data exactly by decompression [13]. This compression is applied to the sound files, pictures or videos.

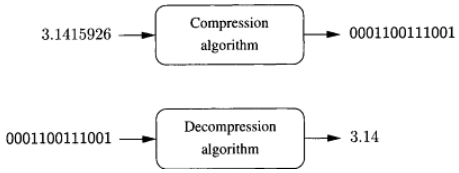


Figure 3. Lossy compression algorithms[13].

2) Shannon-Fano Algorithm

Widiartha [14] and Josua Marinus Silaen [4] in his study, presents the coding technique developed by two people in two different processes, i.e. Cloude Shannon at Bell Laboratory and R.M. Fano at MIT, but because it has a resemblance of the working process then finally this technique is named from the combined of their name. This algorithm is the basic information theory algorithm which is simple and easy to implement [2].

The process of encoding can be done by following the example of string "FARHANNAH". Then do step 1 and 2, resulting in a Table 1 as below:

Table 1. The frequency of the symbol in descending

Symbol	A	H	N	F	R
Total	3	2	2	1	1

After that continued with the creation of a table of codeword Shannon-Fano as below, the steps to make this table simply by following steps 3 and 4.

Table 2. *Codeword* Shannon-Fano

Symbol	Count	step1	step2	step3	code
A	3	0	0		2
H	2	0	1		2
N	2	1	0		2
F	1	1	1	0	3
R	1	1	1	1	3

The table then generate the Shannon-Fano tree is as below:

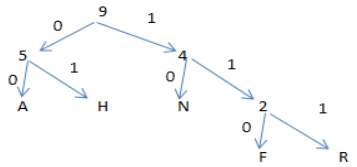


Figure 4. Shannon-Fano Tree

To test the performance of the Shannon-Fanno needed a table containing the results of the performance of the Shannon-Fano, i.e. as follows:

Table 3. Shannon-Fano's performance result

Symbol	Count	code word	# of bits used
A	3	00	6
H	2	01	4
N	2	10	4
F	1	110	3
R	1	111	3

From the table above, note that for the string "FARHANNAH" can be written in binary code 110 00 111 01 00 10 10 00 01 so when encoded into hexadecimal numbers the result is C74A1. Where the total bits needed to write the string "FARHANNAH" after it is compressed using the formula:

$$Weight\ Path = \sum(no.of\ count * no.of\ code)$$ [2]

$$= (3*2)+(2*2)+(2*2)+(1*3)+(1*3)$$

$$= 20$$

Total bits needed after compression is 20 bits, it is more significant in comparison with the needs before compressed, amounting to 72 bits. From the above

calculation, then the resulting compression ratio of 72,22%.

3) Huffman Algorithm

Huffman algorithm was originally introduced by David Huffnan in 1952, where this method is the most popular method in the compression of text. Huffman compression method analyzes in advance against the input string, then it will be processed in the next compression. Huffman tree is created in which a binary tree with optimal replacement code for symbols with a higher probability of occurrence [15]. This algorithm resolves the goal by allowing the symbol length varies. Short code representing a symbol that is often used, and the longer the symbols used to represent that rarely appear in the string[16].

The process of encoding can be done by following the example of the string "FARHANNAH" by making the following frequency table:

Table 4. Character frequencies

Symbol	F	A	R	H	N
Total	1	3	1	2	2

The table above will become fundamental in the creation of a full binary tree. In a study conducted Yellamma and Challa [2]. They present a different way of representing the process of encoding by using the table of possibilities. The following table can be poured with the different examples:

Table 5. The frequency of the symbol in ascending

Symbol	Probablility	step1	step2	step3
F	0.111	0.222	0.222	0.444
R	0.111		0.333	0.556
H	0.222	0.222	0.444	
N	0.222	0.222		
A	0.333	0.333		

Below is the formation of a Huffman tree is retrieved from the table above each symbol emergence prediction:

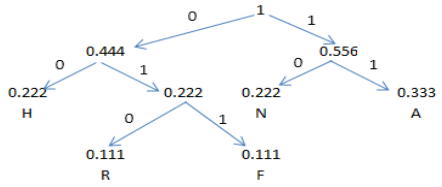


Figure 5. Huffman tree

The resulting code from Huffman algorithm can be calculated the average length per character code. The following is an example of the calculations:

Table 6. Huffman's performance result

Char	count	Probably	codeword	code	# of bits used
F	1	0.111	011	3	3
R	1	0.111	010	3	3
H	2	0.222	00	2	4
N	2	0.222	10	2	4
A	3	0.333	11	2	6

The resulting code from the table above for the string "FARHANNAH" is the 001 11 010 00 11 10 10 11 00 , so if encoded in numbers hexadesimal becomes 3A39C. Where the total bits needed to write the string "FARHANNAH" after it is compressed using the formula:

$$Weight\ Path = \sum(no.of\ count * no.of\ code)[2]$$
$$= (1*3)+(1*3)+(2*2)+(2*2)+(3*2)$$
$$= 20$$

Total bits needed after compression is 20 bits, it is more significant in comparison with the needs before compressed, amounting to 72 bits. From the above calculation, then the resulting compression ratio of 72,22%.

4) LZ77

Lempel-Ziv 77 algorithm (LZ77), also known as LZ1, published in a paper by Abraham Lempel and Jacob Ziv in 1977. This algorithm is lossless algorithm type. LZ77 algorithm is called 'sliding windows', or running windows. This window is divided into two parts, the first part is called the history buffer (H), or search the buffer, containing part of the input characters already encoded. The second window is the look-ahead buffer (L), containing most of the input character will be encoded. Later in its implementation, the history buffer will have a length of a few thousand bytes, and the lookahead buffer length is only tens of bytes [1].

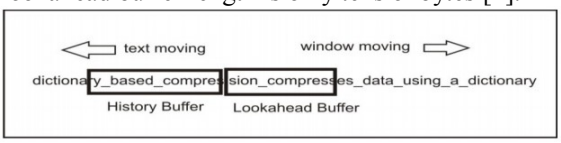


Figure 6. Compression windows of LZ77 [13]

Examples of the application of the making of the token on the LZ77 sequence string "Data_ini_representasi-input_dari_pergeseran_input_pertama_kali_dilakukan". Suppose the sequence has been made on the process of compression to a string "Data_ini_representasi_input_dari_pergeseran". In other words, a sequence of strings that are in the history buffer, the remaining rows will be in the lookahead buffer and data input. Note in the Figure 7, which is the view from the window of LZ77 are applied.

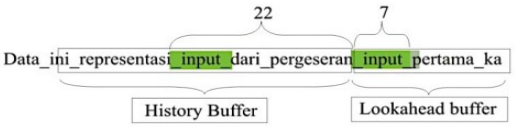


Figure 7. Compression windows of LZ77 [15]

From Figure 7, to be exact on lookahead buffer, there are patterns with a history buffer along the 7 characters, that is a sequence of strings "_input_". This requires that the issue of the value of the token of the specified patterns. Its format will be written like this (22.7, p) of 22 space be offset value, 7 is the length of the pattern and p is the character mismatch. Then do shift window to look like below:

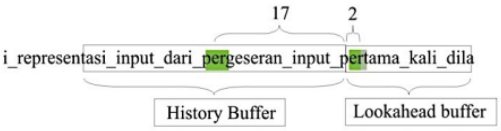


Figure 8. Shifting LZ77's window compression [15]

From figure 7 above looks history buffer and the lookahead buffer shifted along the token pattern found. In the lookahead buffer filled up again as much space, by taking the data from the rest of the input. After that the process of the formation of the token is continued again, by looking at figure 8 and generate tokens (17, 2, t). After that do a shift and the formation of the lookahead buffer until the token contains the end off the file.

5) BWT

Burrows-Wheeler algorithm was introduced in 1994 by Michael Burrows and David Wheeler [8] in a study entitled "A Block-sorting Lossless Data Compression Algorithm". Presented in this study is a data compression algorithm based on a transformation where the bottom line is the discussion method BWT.

Examples to do encoding in the string "FARHANNAH", a step of BTW, as in [13], can be described as follows:

- The rotation (cyclic shift) process on the string S = "FARHANNAH" as much as N-1 times, so that the berordo matrix NxN obtained:

N	0	1	2	3	4	5	6	7	8
0	F	A	R	H	A	N	N	A	H
1	A	R	H	A	N	N	A	H	F
2	R	H	A	N	N	A	H	F	A
3	H	A	N	N	A	H	F	A	R
4	A	N	N	A	H	F	A	R	H
5	N	N	A	H	F	A	R	H	A
6	N	A	H	F	A	R	H	A	N
7	A	H	F	A	R	H	A	N	N
8	H	F	A	R	H	A	N	N	A

Figure 9. Matrix of "FARHANNAH"

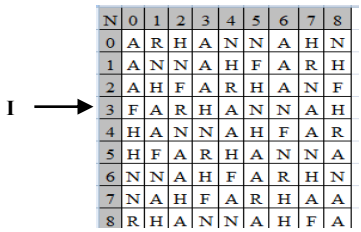
- Sort the results matrix of the rotation in lexicographic matrix rows on:

N	0	1	2	3	4	5	6	7	8
0	A	R	H	A	N	N	A	H	N
1	A	N	N	A	H	F	A	R	H
2	A	H	F	A	R	H	A	N	F
3	F	A	R	H	A	N	N	A	H
4	H	A	N	N	A	H	F	A	R
5	H	F	A	R	H	A	N	N	A
6	N	N	A	H	F	A	R	H	N
7	N	A	H	F	A	R	H	A	A
8	R	H	A	N	N	A	H	F	A

Figure 10. Matrix, the string "FARHANNAH" after sorted in lexicographi.

- Based on the above image retrieved string L formed from the last character in each row of the matrix, and the index I stating the position of the original string, so that the results of encoding of string S = "FARHANNAH" stated in (L, I) is (NHFHRANAA, 3).

L



N	0	1	2	3	4	5	6	7	8
0	A	R	H	A	N	N	A	H	N
1	A	N	N	A	H	F	A	R	H
2	A	H	F	A	R	H	A	N	F
3	F	A	R	H	A	N	N	A	H
4	H	A	N	N	A	H	F	A	R
5	H	F	A	R	H	A	N	N	A
6	N	N	A	H	F	A	R	H	N
7	N	A	H	F	A	R	H	A	A
8	R	H	A	N	N	A	H	F	A

Figure 11. The results matrix encoding string "FARHANNAH".

The process of decoding BWT, requires a pair (L, I) that is used to create the string S along the N character, the following steps:

- Establish the first character of rotation
  These measures form the first column of the matrix F M, where the formation of matrix F can be done with a sort of character string L = "NHFHRANAA" so that the retrieved F = "AAAFHHNNR".
- Reshaping the string S.
  This stage contains the step to the re-establishment of a string S according couple (L, I) = (NHFHRANAA, 3) and F = string "AAAFHHNNR". The index I is the key to reshaping the string S, because the index I as the first character of the string pointer S.

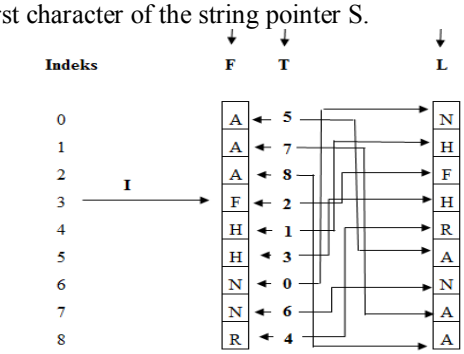


Figure 12. The process of the formation of string S based on a string of L, F and index I.

This process produces one vector transformation T where each element indicates the mapping from string to string element F L on an ongoing basis. This is done continuously until the entire element is successfully mapped, so then this process produces a string S that rearrange.

6) MTF

MTF is a transformation algorithm which does not perform data compression but may help reduce redundancies, such as the result of the BWT transformation [9]. The basic idea of this method is to maintain the alphabet A of symbols as a list where frequently occurring symbols are located near the front [1].

In the encoding process, the technique used is to encode a stream of symbols based on code adaptation. A symbols "**s**" is encoded as the number of symbol that precede it in this list. Thus if A =(t,h,e,s,..) and the next symbol in the input stream to be encoded is "e", it will

be encoded as 2, since it is preceded by two symbols. After symbol "s" is encoded, it is moved to the front of list A. Thus, after encoding "e'', the alphabet is modified to A=(e,t,h,s,..) [1].

To represent annotations, taken the example list alphabetical order A to be used as a reference in coding, i.e. A = (A, F, H, N, R) and the input stream is "FARHANNAH". The following table is an illustration of the process of encoding takes place.

Table 7. MTF encoding process

String	Symbol					Encode
	0	1	2	3	4	
FARHANNAH	A	F	H	N	R	
ARHANNAH	F	A	H	N	R	1
RHANNAH	A	F	H	N	R	11
HANNAH	R	A	F	H	N	114
ANNAH	H	R	A	F	N	1143
NNAH	A	H	R	F	N	11432
NAH	N	A	H	R	F	114324
AH	N	A	H	R	F	1143240
H	A	N	H	R	F	11432401
	H	A	N	R	F	114324012

Based on that table, the result is C=(114324012).

B. Combination of Compression Algorithms

The combination of BWT + Distance Coding (DC) + Fibonacci Coding was proposed by [7] which implemented on a text file. The advantages of DC from MTF was its ability to reduce the length of the input; however the drawback is that it generates a huge alphabet. Fibonacci very well adapt to DC for FC is a universal code.

The study on performing testing of the LZW compression algorithm, Huffman, a Fixed-Length Code (FLC), and combined FLC + Huffman (HFLC) on the United Kingdom speaking text file types, from the Calgary Corpus and some private sources researchers [17]. The result of the LZW compression algorithm being the most efficient in this study.

The compression techniques was studied and tested based on dictionary and statistical-based text files [18]. The statistical based compression technique that were examined, among others, Arithmetic, Adaptive Huffman, Huffman, Shnnon-fano. As for the dictionary-based compression technique, namely: LRE, LZB, LZ77, LZH, LZSS, LZW, LZFG, LZR, LZC, LZT. The results obtained have the most efficient level of Arithmetic to statistical compression techniques, followed by adaptive Huffman, then Huffman, and Shannon-fano and RLE. The dictionary-based compression technique, LZB exceed LZ77, LZH, LZSS, LZR.

The analysis algorithm for transformation of Length-Preserving Transform Index (LIPT) and its derivatives as well as Star-New Transform (StarNT) was studied in [6]. All of the transformations are then combined with Bzip2, Gzip, PPMD. The result is LIPT + Bzip2 is much faster in performance time. StarNT works better than LIPT when applied with the backend compressor.

The application of reversible transformations to the source file before applying compression algorithms was studies in [19]. Careful transformation algorithm is Reinforced Intelegent Dictionary Base Encoding (RIDBE). Then the proposed algorithm combination i. e. RIDBE + BWT + MTF + RLE + Arithmetic. The result is a significant increase in text data compression.

The analysis of benefits/advantages and influence in lossless compression using BWT transformation algorithm Intelegent Dictionary Based Encoding (IDBE), Enhance Intelegent Dictionary Based Encoding (EIDBE) and Improved Intelegent Dictionary Based Encoding (IIDBE) as a method of preprocessing in text files obtained from the Calgary corpus [20]. These combinations were tested with the following algorithms: PKZIP, BWA, *-enc$_+$ BWA, IDBE+ BWA, EIDBE + BWA and IIDBE + BWA. The result is interesting, it significantly increases the compression of text data. IIDBE indicates an increased of 18,32% more over BWA and an increases of 8,55% compared to the BWA + *-encode, an increase of 2.28% over IDBE + BWA and an increase about 1% more than IEDBE + BWA.

The compression algorithms of Shannon-Fano, Huffman, LZW and LRE were also compered on text files as reported in [2]. The results obtained that the Shannon-Fano compression was resultingat nearly the same with Huffman compression which save about 54,7% space.

The method of front-end and back-end in the process of compression was studied in[21]. The front-end method proposed to use an algorithm of transformation Star encoding prior to compression by arithmetic, Huffman, PPM and BWA on text files from the Calgary Corpus, in English language patterns, and Spain, France, Germany as well. The results show that the English language is the language that is most sensitive to this algorithm which generates the highest performance of 18% compared to coding arithmetic. Insensitive language is Spain, as the lowest performance improvements i.e. 0.21% on combined *-Enc. + PPM algorithm.

Joint transformation algorithm with compression was studied in [11]. The proposed algorithm combined the following: Method1: BWT + RLE + MTF + RLE + Huffman, Then, the authors develop Method2 = Dictionary + Method1. Dictionary algorithm used is StarNT. While the comparison algorithm are BWCA, *Dicitionary Method1= Dictionary* + BWCA. The results obtained on average size of compressed files showed that the method of Dictionary + BWCA had high results (7.180,3 bytes) than Dictionary + Proposed method (7453,1 byte) where the original file size of 15.452,3 byte.

The other research conduct a test in detail to the LZ77 algorithm, Huffman, a combination of LZ77 + Huffman and Deflate on text files with a specific pattern[15]. The result of this study is that a deflate algorithm which has the best performance of Huffman, LZ77, LZ77 + Huffman, LZ77

static + dynamic Huffman, deflate compression ratio up to 38.84%.

To get accurate results, this study discusses a combination of transformation statistical data compression algorithms which used intensive hardware and software as required.

III. RESEARCH METHODOLOGY

A. Research methods

This study employed an experimental method on combination of BWT + MTF + Shannon-Fano algorithms. Then the result from the combined algorithm was compared to a single compression algorithm, Shannon-Fano, Huffman and LZ77. Each experiment was done using a text file as the object file. Several experiments were conducted in the studyI which ncludes the following:

- Experiment: Input
  The parameters used in this study is the size of the file. The file size is expressed in unit of bytes (consisting of 8 bits), the file types used are files with extension of txt, doc, and rtf.
- Experiment: Process
  In this part, process was done 3 times on each test sample file for each algorithm, such as the combined algorithm of BWT + MTF + Shannon-Fano and 3 single compression algorithms, Shannon-Fano, Huffman and LZ77. This process includes the encoding process (process text file compression) and decoding (the process of reversing compressed files to be back to its original state).
- Experiment: Output.
  The output is the end result in a file with the extension in accordance with the algorithm that is used. On the implementation of the Shannon-Fano, it will generate the file with .sf extention, Huffman with .huff extention, LZ77 with .lz extention and the transformation of BWT + MTF will generate a file with .bm extension, while the combined algorithm of BWT + MTF + Shannon-Fano will produce a file with .bms extension.

B. Data Collection

The data used as a sample in testing this compression algorithm is a text file that has the extension of doc, txt, and rtf. This study used 10-sample ASCII text files, 5 files are of the Calgary Corpus, namely: bib, book2, news, and trans progc. While 5 other files are the author's development file contains the combination of other files from the files of the Calgary Corpus, namely: test1, test2, …, test5.

Tabel 8. File sample

No	File Name	Size(Bytes)		
		.txt	.doc	.rtf
1	Bib	111261	247808	251932
2	book2	610856	921088	1318946
3	News	377109	583680	824165
4	Progc	39611	89088	136903
5	Trans	93695	165888	254746
6	Test1	3132450	4514816	6298394
7	Test2	1131199	1708032	2395628
8	Test3	1475157	2232320	3141306
9	Test4	1879470	2717184	3818931
10	Test5	3871680	5641728	7428047

C. Analysis Techniques

Several methods of analysis were applied in this study. The first analysis techniques is an analysis process on how it works as well as the analysis of algorithms on the 5 types of compression algorithms used, i.e., BWT, MTF, Shannon-Fano, Huffman and LZ77. The second is the technique of analysis of processes/procedures as well as the analysis of the algorithm from a combination of algorithms that will be examined, the combination of these algorithms is BWT + MTF + Shannon-Fano. The third is the last analysis of the test process i.e. the data measurement from the sample files provided to conduct a performance evaluation of algorithms.

Each test group consists of 10 different files and has the following variables:

- File Size
  The size of the files used in this research is expressed in units of bytes. The size of each file is recorded based on the name that goes through the use and analysis of the size of the file which made up of two sizes, i.e. the size of the original file size and the size after the compression.
- Ratio
  Calculation of the ratio is the result of a reduction of 100 with the quotient of the compressed file size of original size then the result is multiplied by 100.

$$\text{Ratio} = 100 - \left(\frac{Compressed\ file\ size}{Original\ file\ size}\right)\ x\ 100$$

- Runtime/ compression time
  Compression time is the time to do the compression algorithm on each symbol or character (byte) in the original file which is expressed in units of milliseconds/millisecond (m/s). The time retrieved from early time calculation end time reduced compression compression on a specific allocation memory.

$$Compression\ time = Starting\ time - Finishing\ time$$

In summary, the data analysis techniques used in this research can be seen from the diagram depicted in Figure 13

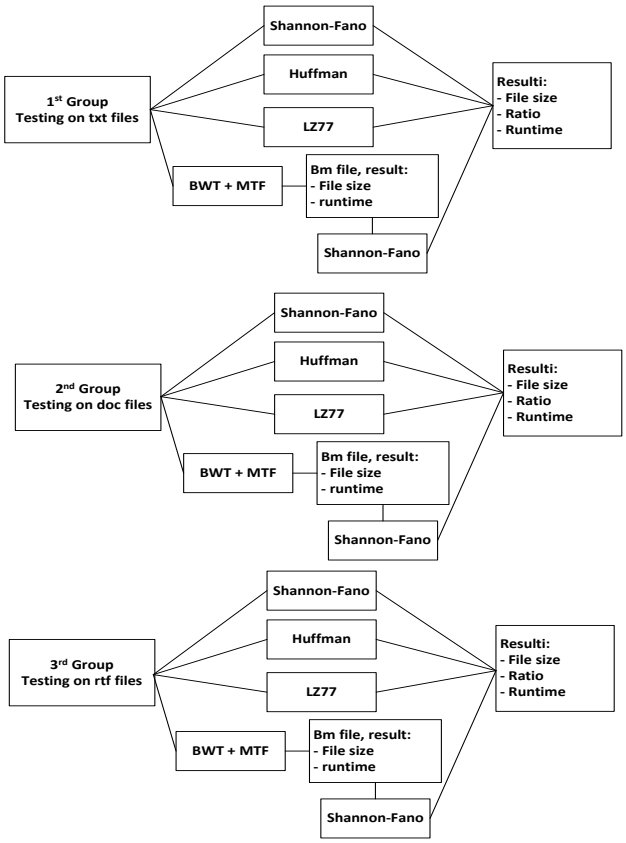


Figure 13. Data analysis methods used in the study

IV. FINDINGS AND DISCUSSION

A. Findings from the Experiments

1) Files with extension .txt

Tabel 9. Comparison results of file size compression on txt files

No	File Name	File Size (bytes)			
		Shannon-Fano	Huffman	LZ77	BMS
1	bib.txt	73773	72862	52514	53924
2	book2.txt	375415	368420	319184	287850
3	news.txt	248532	246516	208406	197178
4	progc.txt	26321	26029	20608	17083
5	trans.txt	64196	63876	27577	36016
6	test1.txt	1941646	1908295	1601746	1463710
7	test2.txt	745249	724476	584611	547796
8	test3.txt	971274	960170	721654	695293
9	test4.txt	1165046	1145025	962410	878303
10	test5.txt	2568026	2548143	2086827	2005751

LZ77 gives the best result in compressing file trans.txt, the original size is 93695 bytes, it compressed to 27577 bytes, means the total reduction is 66118 bytes. The combination of BWT + MTF + Shannon-Fano algorithm compressed to 36016 bytes. However on a file progc.txt, the result from LZ77 is worst than the combination of BWT + MTF + Shannon-Fano algorithm, i.e. 17083 bytes of size 39611 bytes. On average the smallest compressed file size by an algorithm combination of BWT + MTF + Shannon-Fano, is 618290 bytes. While the Shannon-Fano algorithm and Huffman are less performed compared to the other 2 algorithms .

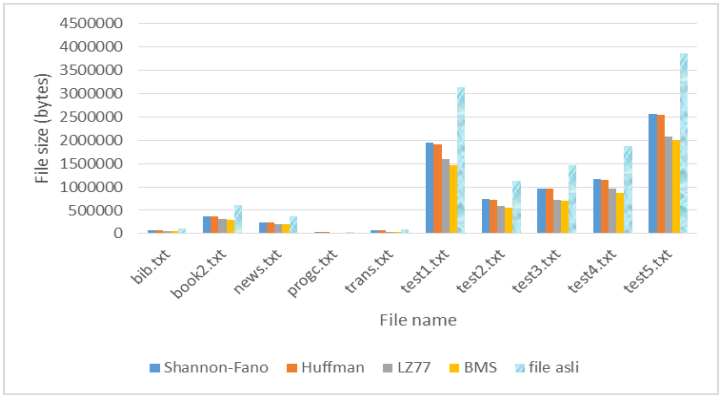


Figure 14. Comparison of the original file size and compression results

Tabel 10. Comparison of compression ratio on txt files

No	File Name	Ratio (%)			
		Shannon-Fano	Huffman	LZ77	BMS
1	bib.txt	33.69	34.51	52.8	51.58
2	book2.txt	38.54	39.69	47.75	52.92
3	news.txt	34.1	34.63	44.74	52.2
4	progc.txt	33.55	34.29	47.97	56.92
5	trans.txt	31.48	31.48	70.57	61.6
6	test1.txt	38.02	39.08	48.87	53.32
7	test2.txt	34.12	35.96	48.32	51.62
8	test3.txt	34.16	34.91	51.08	52.91
9	test4.txt	38.01	39.08	48.79	53.31
10	test5.txt	33.67	34.19	46.1	48.24

LZ77 produced a ratio of 70.57 % to trans.txt file. On average the combination of BWT + MTF + Shannon-Fano has the highest compression ratio than LZ77, i.e. 53.46% while the LZ77 is 50,7%. Huffman and Shannon-Fano has shown a lower performance than the two previous algorithms, but the average ratio of Huffman is better than Shannon-Fano.

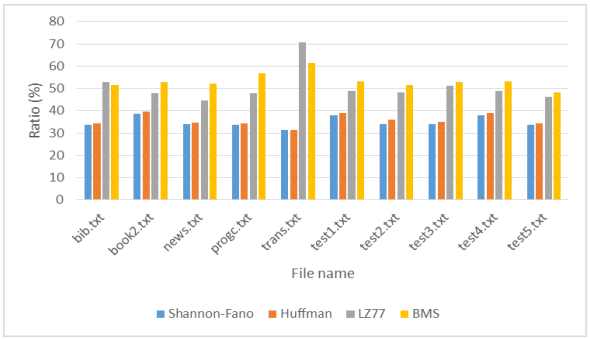


Figure 15. Comparison of compression ratio on txt files.

Tabel 11. Comparison of encoding time on txt files.

No	File Name	Encoding time (ms)			
		Shannon-Fano	Huffman	LZ77	BMS
1	bib.txt	7.29	6.43	1328.58	52.17
2	book2.txt	33.42	33.55	13646.77	189.17
3	news.txt	22.56	21.75	10771.60	136.75
4	progc.txt	2.67	2.33	7035.95	22.92
5	trans.txt	5.94	5.39	604.29	53.20
6	test1.txt	101.09	103.45	71221.79	757.89
7	test2.txt	54.86	47.81	28403.74	300.12
8	test3.txt	52.52	63.20	35315.62	391.38
9	test4.txt	62.00	71.43	42336.95	462.79
10	test5.txt	127.38	120.02	119180.92	901.73

In the table above, it can be seen that the best time in processing is held by Huffman on progc.txt i.e. 2.33 ms, while Shannon Fano in the second position with 62 ms; the difference of 9,43 ms from Huffman. This makes the Shannon-Fano has the lowest compression process time in average than all the other algorithms examined, followed by Huffman, the combined algorithm of BWT + MTF + Shannon-Fano and then LZ77 with the highest level of processing time.

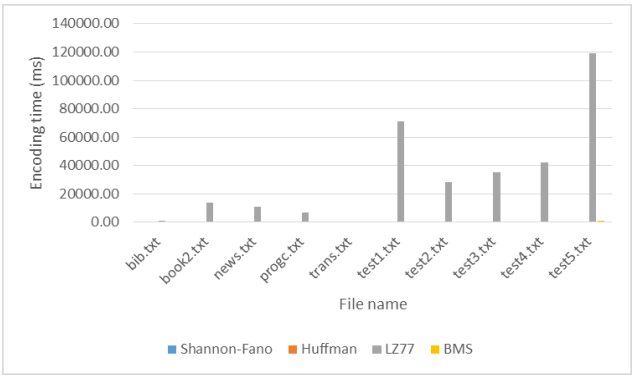


Figure 16. Comparison of encoding time on txt files.

2) Files with extension doc

Table 12. Comparison results file size compression on doc files.

No	File Name	File size (bytes)			
		Shannon-Fano	Huffman	LZ77	BMS
1	bib.doc	146056	140136	87784	88915
2	book2.doc	575005	561628	400654	365521
3	news.doc	373577	370978	262778	253185
4	progc.doc	52840	51003	33706	32764
5	trans.doc	103277	101823	46405	56551
6	test1.doc	2825994	2767716	1966967	1791126
7	test2.doc	1078407	1065326	736607	696555
8	test3.doc	1421340	1407421	921497	882182
9	test4.doc	1702945	1667211	1183438	1078525
10	test5.doc	3717773	3620930	2548398	2444488

Results in Table 12 show that for the bib.doc and trans.doc, LZ77 produces smaller file size compared to the combined algorithms. However in other files, a combination of BWT + MTF + Shannon-Fano produces smaller files than that produced by LZ77 with a difference of 49842 bytes.

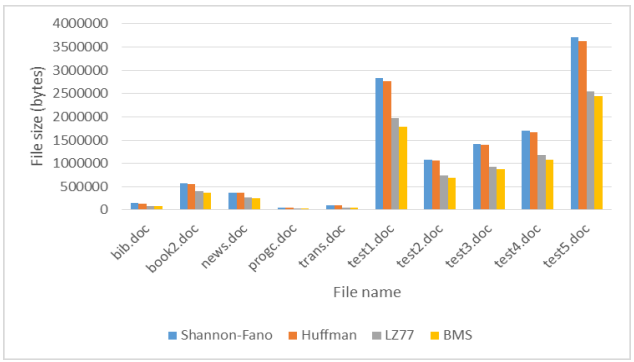


Figure 17. Comparison of the original file size and compression results on doc files

Tabel 13. Comparison of compression ratio on doc files.

No	File Name	Ratio (%)			
		Shannon-Fano	Huffman	LZ77	BMS
1	bib.doc	41.06	43.45	64.58	64.15
2	book2.doc	37.57	39.03	56.5	60.36
3	news.doc	36	36.44	54.98	56.67
4	progc.doc	40.69	42.75	62.17	63.36
5	trans.doc	37.74	38.62	72.03	65.94
6	test1.doc	37.41	38.7	56.43	60.37
7	test2.doc	36.86	37.63	56.87	59.26
8	test3.doc	36.33	36.95	58.72	60.52
9	test4.doc	37.33	38.64	56.45	60.35
10	test5.doc	34.1	35.82	54.83	56.71

Data in Table 13 show that the highest compression ratio of 72,03% is produced by LZ77 on file trans.doc. For all samples, the difference on compression ratio between LZ77 and the combined-algorithm reach up to 1.41%. While Huffman and Shannon-Fano shows lower compression ratio than LZ77 and combination-algorithm. Huffman algorithm performs better than the Shannon-Fano, which shows a difference of approximately 1.29%.

Figure 18. Comparison of compression ratio on doc files.

Tabel 14. Comparison of encoding time on doc files.

No	File Name	Encoding time (ms)			
		Shannon-Fano	Huffman	LZ77	BMS
1	bib.doc	12.47	10.54	4653.81	630.49
2	book2.doc	37.37	42.31	21016.20	1278.98
3	news.doc	32.40	28.09	15674.75	1064.70
4	progc.doc	4.99	4.79	983.50	481.55
5	trans.doc	9.64	9.42	2312.74	509.33
6	test1.doc	138.93	140.73	107750.02	5143.49
7	test2.doc	67.29	69.51	43599.83	2262.86
8	test3.doc	89.83	85.54	55260.01	2914.48
9	test4.doc	104.38	97.10	64331.59	3150.51
10	test5.doc	170.96	186.17	167618.64	6340.34

The most efficient compression time generated by Huffman is 4,79 ms, which is 0.2 ms faster than the Shannon-Fano on the file progc.doc which is the smallest file. The combination of BWT + MTF + Shannon-Fano takes longer than two previous algorithms, about 481,55 ms, while the LZ77 is the slowest, about 983,5 ms on the same file.

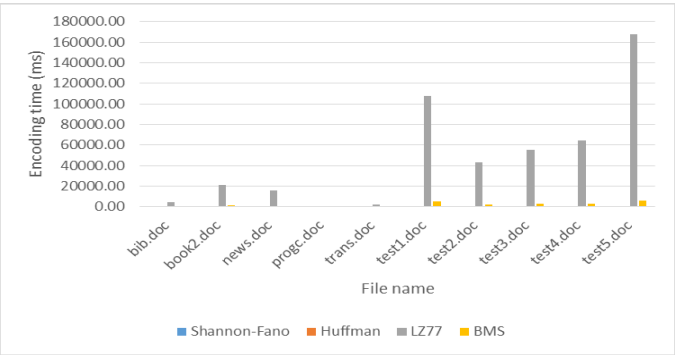


Figure 19. Comparison of encoding time on doc files.

3) Files with extension rtf

Table 15. Comparison results file size compression on rtf files

No	File Name	File Size (bytes)			
		Shannon-Fano	Huffman	LZ77	BMS
1	bib.rtf	170409	168342	83120	84540
2	book2.rtf	812038	7965597	391716	409287
3	news.rtf	536503	527205	265053	286081
4	progc.rtf	88434	86962	36250	38637
5	trans.rtf	167207	163537	46937	71624
6	test1.rtf	3840503	3743113	1830575	1950773
7	test2.rtf	1498529	1471264	683697	765308
8	test3.rtf	1981980	1942279	838231	974243
9	test4.rtf	2336876	2276659	1109844	1181813
10	test5.rtf	4745781	4661040	2363535	2620748

For all sample files with an extension .rtf, LZ77 shows better performance compared to three other algorithms. The combined algorithm of BWT + MTF + Shannon-Fano shows bigger file size of 73410 bytes compared to LZ77, followed by Huffman and Shannon-Fano.

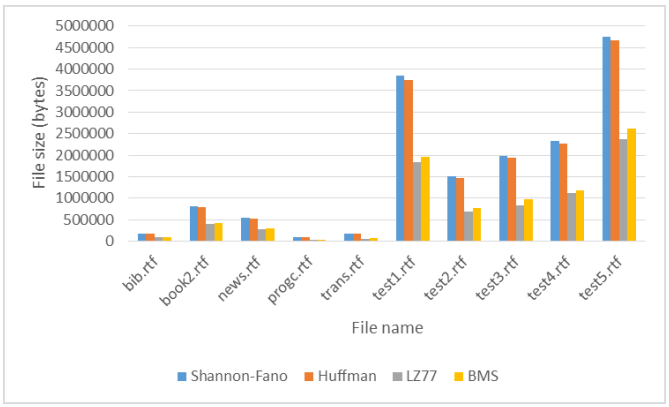


Figure 20. Comparison of the original file size and compression results on rtf files.

Table 16. Comparison of encoding time on rtf files.

No	File Name	Ratio (%)			
		Shannon-Fano	Huffman	LZ77	BMS
1	bib.rtf	32.36	33.18	67.01	66.48
2	book2.rtf	38.43	39.6	70.3	69
3	news.rtf	34.9	36.03	67.84	65.32
4	progc.rtf	35.4	36.48	73.52	71.81
5	trans.rtf	34.36	35.8	81.57	71.91
6	test1.rtf	39.02	40.57	70.94	69.06
7	test2.rtf	37.45	38.59	71.46	68.09
8	test3.rtf	36.91	38.17	73.32	69.02
9	test4.rtf	38.81	40.38	70.94	69.08
10	test5.rtf	36.11	37.25	68.18	64.75

LZ77 shows the highest compression ratioa, which is 3.06% higher than the combination of BWT + MTF + Shannon-Fano. While Huffman and Shannon-Fano show compression ratios less than 50%, where Huffman is a little of 1.23% higher than the Shannon-Fano.

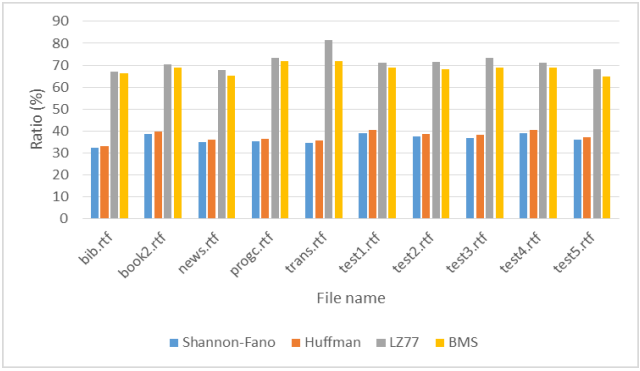


Figure 21. Comparison of compression ratio on rtf files.

Table 17. Comparison of encoding time on rtf files.

No	File Name	Encoding time(ms)			
		Shannon-Fano	Huffman	LZ77	BMS
1	bib.rtf	13.52	13.16	3606.98	123.58
2	book2.rtf	62.33	56.35	16998.34	536.49
3	news.rtf	38.97	40.52	14278.82	336.97
4	progc.rtf	7.16	6.92	1385.31	99.08
5	trans.rtf	14.16	12.99	1902.15	160.24
6	test1.rtf	159.65	154.98	79325.66	2403.34
7	test2.rtf	89.30	84.13	32689.37	963.74
8	test3.rtf	105.35	109.84	39923.27	1295.60
9	test4.rtf	121.43	108.75	48026.79	1498.19
10	test5.rtf	203.41	207.29	130549.38	2676.35

LZ77 is the algorithm that consume the highest compression time, which can be presented in Figure 22 which shows that LZ77 compression time dominates the encoding time compared to the other algorithms.

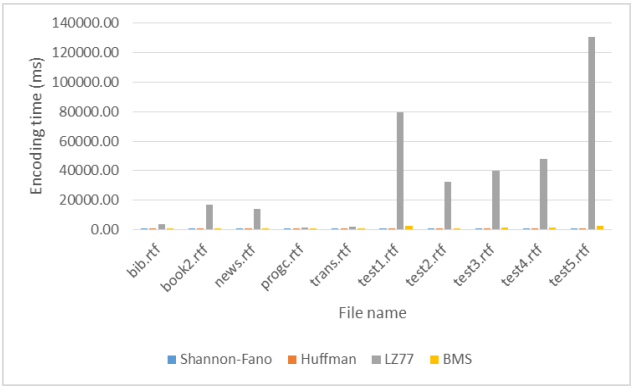


Figure 22. Comparison of encoding time on rtf files.

B. Results Analysis

1) File Size

Table 18. The average size of compressed files in each test file type

Test file type	Average of file size (bytes)				
	Original	Shannon-Fano	Huffman	LZ77	BMS
Txt	1272249	817947.8	806381.20	658553.7	618290.4
Doc	1882163	1199721.4	1175417.20	818823.4	768981.2
Rtf	2586900	1617826	1583699.80	764895.8	838305.4

Table 18 presents the result of compression process using several compression alogorithm. The combination of BWT + MTF + Shannon-Fano algorithm has shown better prformance than LZ77, Huffman or Shannon-Fano for the sample files used.

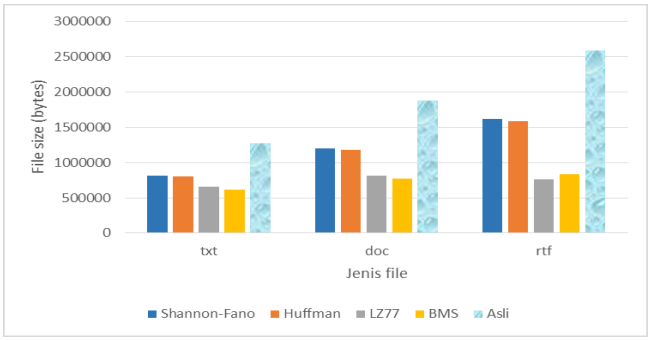


Figure 23. Graphic analysis of testing the size of the compressed files for each file type sample test.

Table 19 presents the average file size of compressed sample files on the overall test.

Table 19. The results of the analysis of compressed file size testing

Type	N	Average of file size (bytes)			
		Min	Max	Sum	Avg
Original	30	39611	7428047	57413118	1913770.6
Shannon-Fano	30	26321	4745781	36354952	1211831.7
Huffman	30	26029	4661040	35654982	1188499.4
LZ77	30	20608	2548398	22422729	747424.3
BMS	30	17083	2620748	22255770	741859

Figure 24. presents the percentage of compressed file size on the overall test files.

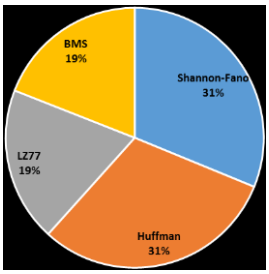


Figure 25. Analysis percentage of compressed file size

2) Compression Ratio

Table 20 The average compression ratio for each file type test

Test file type	Average of ratio (%)			
	Shannon-Fano	Huffman	LZ77	BMS
Txt	34.934	35.78	50.699	53.462
Doc	37.509	38.80	59.356	60.769

rtf	36.375	37.61	71.508	68.452

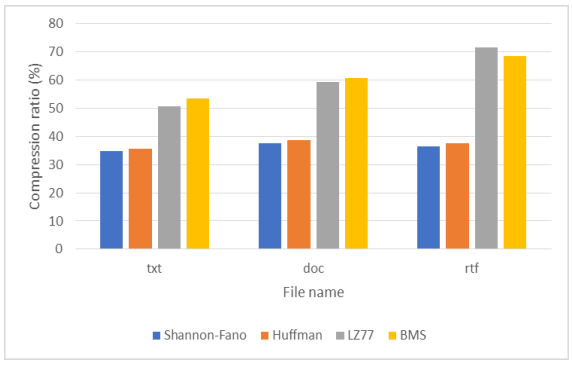


Figure 26. Graph the average compression ratio.

For the testing on file types .txt and .doc the combination of BWT + MTF + Shannon-Fano shows a higher compression ratio than other algorithms, on testing file .rtf LZ77 performs better. This makes the combined algorithm of BWT + MTF + Shannon-Fano produce more efficient compression ratio than LZ77, while Huffman and Shannon-Fano shows lower performance, eventhough Huffman performs 1% higher compare to the Shannon-Fano.

Table 21. Analysis of the compression ratio

Type	N	Average of ratio compression (bytes)			
		Min	Max	Sum	Avg
Shannon-Fano	30	31.48	41.06	1088.18	36.27
Huffman	30	31.48	43.45	1121.90	37.40
LZ77	30	44.74	81.57	1815.63	60.52
BMS	30	48.24	71.91	1826.83	60.89

Combination of BWT + MTF + Shannon-Fano has a 0.37% higher compression ratio than the LZ77, while Huffman is 1, 2% higher than the Shannon-Fano.

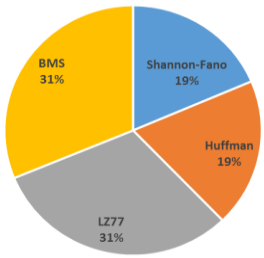


Figure 27. Graph the percentage of compression ratio analysis

3) Compression Time

Table 22. Average time compression on each type of sample files.

Test file type	Average of compression time (ms)			
	Shannon-Fano	Huffman	LZ77	BMS
Txt	46.97	47.54	32984.62	326.81
Doc	66.83	67.42	48320.11	2377.67
rtf	81.53	79.49	36868.61	1009.36

LZ77 performs in a fairly high figure compared to the other algorithms. The chart in Figure 28 shows the dominance of the compression time of LZ77 compared to other algorithms.

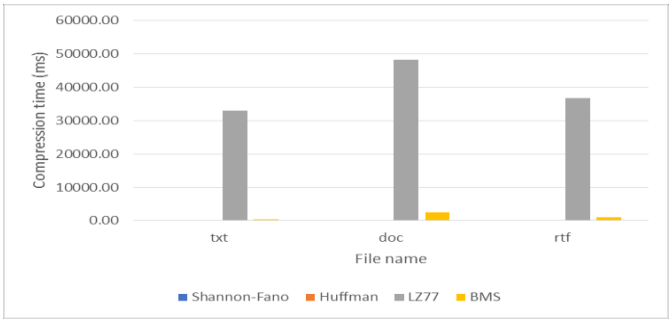


Figure 28. Comparison of time compression on each type of file.

Figure 29 shows that the LZ77 algorithm has the highest compression time compared to others. Table 23 presents the average compression time of all files.

Table 23. Average compression time on all sample files.

Type	N	Average of compression time (ms)			
		Min	Max	Sum	Avg
Shannon-Fano	30	2.67	203.41	1953.26	65.11
Huffman	30	2.33	207.29	1944.49	64.82
LZ77	30	604.29	167618.64	1181733.36	39391.11
BMS	30	22.92	6340.34	37138.44	1237.95

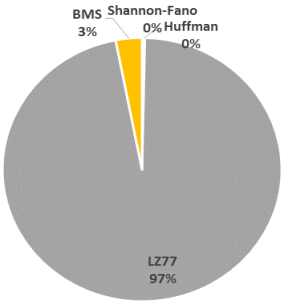


Figure 29. The percentage of compression time on all sample files.

C. Research Implications

Table 24. Average analysis testing on all test files.

Aspect	Shannon-Fano	Huffman	LZ77	BMS
File Size	63.32	62.10	39.06	38.76
Ratio	36.27	37.40	60.52	60.89
Compression time	65.11	64.82	39391.11	1237.95

Table 24 presents an overview of algorithms which their lowest compression performance on each aspect. This supports the previous research that the implementation of the Shannon-Fano will result in less efficient compression

performance compared to the implementation of Huffman, although both have the similar compression process.

Test analysis results show that the most efficient compression ratio produced by the combined algorithm of BWT + MTF + Shannon-Fano is 60,89%, while the LZ77 is 60,52%, and Huffman and Shannon-Fano produces a ratio of less than 50%.

On the aspect of compression time, it has been known that LZ77 needs a very long time to compress, which is about 39391,11 ms. It equals to more than 39 seconds, or about 0.65 minutes or approximately 96.63% slower than BMS. This take a long time to process using a considerably good hardware specifications used in this study. By using less hardware specifications, it will take longer processing time. The combination of BWT + MTF + Shannon-Fano takes 1237,95 ms or approximately 3.04% slower than Shannon-Fano, 18% slower compared to the rest, and around 0,16% slower compared to Huffman.

From compressed file size and ratio aspects, the combination of BWT + MTF + Shannon-Fano and LZ77 show almost the same performance. However, the LZ77 compression time takes the longest. This aspect is important, as the implementation of the algorithm must consider the availability on hardware specification aspect, and speed will also affected by the performance of the CPU and the size of memory. Thus, the combination of BWT + MTF + Shannon-Fano is considered as the most optimal algorithm in this study.

V. CONCLUSION

The implementation of the Shannon-Fano for compressing text file has shown less efficient compression performance compared to Huffman compression, even-though both have similat way of compression process. The average of file size of Shannon-Fano is 63,32% higher than Huffman (with a difference about 1.22%). This affects the average compression ratio, where Shannon-Fano reach 36,27% lower than Huffman with 1.12% difference. The average of compression time of Shannon-Fano is higher than Huffman, which is 65,11 ms compared to 64,82 ms repectively.

In conclusion, the most efficient compression ratio produced by the combination of BWT + MTF + Shannon-Fano which is 60,89%, compared to LZ77 with 60,52% of the ratio. On the compression time, the LZ77 is the slowest, it took about 39391,11 ms, while the combination of BWT + MTF + Shannon-Fano algorithm takes 1237,95 ms. From all aspects, the combination of BWT + MTF + Shannon-Fano algorithm performs better compared to LZ77.

The following works can be considered as further studies in the compression algorithm area:

- the next research work can explore the study by adding other algorithms, such as RLE prior to implementation of the Shannon-Fano.
- Implementation of the combination of BWT + MTF + Shannon-Fano algorithm on different type of files, other than text, such as images, audio and video.

REFERENCES

[1] D. Solomon, *Data Compression*, 4th ed. London: Spriger, 2007.

[2] P. Yellamma and N. Challa, "Performance Analysis Of Different Data Compression Techniques On Text File," *Int. J. Eng. Res. Technol.*, vol. 1, no. 8, (Oktober, pp. 1–6, 2012.

[3] B. Souley, P. Das, and S. Tanko, "A Comparative Analysis of Data Compression Techniques," *Int. J. Appl. Sci.*, vol. 2, no. 10, (April, pp. 63–82, 2014.

[4] J. M. Silaen, "Studi Perbandingan Algoritma Huffman dan Shannon- Fano dalam Pemampatan File Teks," *Pelita Inform. Budi Darma*, vol. 7, no. 1, (Juli, pp. 60–66, 2014.

[5] K. Rastogi, K. Sengar, and M. T. Scholar, "Analysis and Performance Comparison of Lossless Compression Techniques for Text Data," *Int. J. Eng. Comput. Res.*, vol. 2, no. 1, pp. 16–19, 2014.

[6] P. Jeyanthi and V. Anuratha, "Analysis of Lossless Reversible Transformation Algorithms to Enhance Data Compression," *J. Glob. Res. Comput. Sci.*, vol. 3, no. 8, (Agustus, pp. 56–62, 2012.

[7] R. Bastys, "Fibonacci Coding Within the Burrows-Wheeler Compression Scheme," *Electron. Electr. Eng.*, vol. 1, pp. 28–32, 2010.

[8] M. Burrows and D. Wheeler, "A Block-Sorting Lossless Data Compression Algorithm," *Algorithm, Data Compression*, no. 124, p. 18, 1994.

[9] A. S. E. Campos, "Move to Front," 1999. [Online]. Available: http://www.arturocampos.com/ac_mtf.html. [Accessed: 15-Mar-2017].

[10] M. Dipperstein, "Burrows-Wheeler Transform Discussion and Implementation," 2010. [Online]. Available: http://michael.dipperstein.com/bwt/. [Accessed: 15-Mar-2017].

[11] R. R. Baruah, V. Deka, and M. P. Bhuyan, "Enhancing Dictionary Based Preprocessing For Better Text Compression," *Int. J. Comput. Technol.*, vol. 9, no. 1, (Maret, pp. 4–9, 2014.

[12] R. Râdescu, "Transform Methods Used in Lossless Compression of Text Files," *Rom. J. Inf. Sci. Technol.*, vol. 12, no. 1, (November, pp. 101–115, 2009.

[13] I. M. Pu, *Fundamental Data Compression*. London: Butterworth-Heinemann, 2006.

[14] P. Widhiartha, "Pengantar Kompresi Data (Introduction to Data Compression)" 2008, Available: http://www.ilmukomputer.org/ wp-content/uploads/2008/10/ widhiartha_kompresidata.pdf. [Accessed: 1-Nov-2017].

[15] C. P. Nugraha and R. G. Santosa, "Perbandingan Metode LZ77, Metode Huffman dan Metode Deflate Terhadap Kompresi Data Teks," *Informatika*, vol. 10, no. 2, pp. 80–91, 2014.

[16] M. Dipperstein, "Huffman Code Discussion and Implementation," 2010. [Online]. Available: http://michael.dipperstein.com/Huffman/index.html. [Accessed: 16-Apr-2017].

[17] H. Altarawneh and M. Altarawneh, "Data Compression Techniques on Text Files: A Comparison Study," *Int. J. Comput. Appl.*, vol. 26, no. 5, (Juli, pp. 42–54, 2011.

[18] S. Shanmugasundaram and R. Lourdusamy, "A Comparative Study Of Text Compression Algorithms," *Int. J. Wisdom Based Comput.*, vol. 1, no. 4, (December, pp. 68–76, 2011.

[19] S. Senthil, S. J. Rexiline, and L. Robert, "RIDBE: A Lossless, Reversible Text Transformation Scheme for Better Compression," *Int. J. Comput. Appl.*, vol. 51, no. 12, (Agustus, pp. 35–40, 2012.

[20] P. Jeyanthi and V. Anuratha, "Bwt Based Lossless Reversible Transformation Algorithms – An Analysis," *Int. J. Eng. Res. Appl.*, vol. 2, no. 5, (September-Oktober, pp. 807–814, 2012.

[21] E. C. Cankaya and O. Darwish, "Improving Compression Performance with a Star Encoding Front End : A Linguistic Comparison," Proceedings of the International Conference on Foundations of Computer Science (FCS); Athens : 1-7 2013.