Comparative analysis of the Compression of text data using Huffman, Arithmetic,

Run-Length, and Lempel Ziv Welch coding algorithms

Abstract

The purpose of the study was to compare the compression ratios of file size, file complexity, and time used in compressing each text file in the four selected compression algorithms on a given modern computer running Windows 7. The researcher used the Java programming language on the NetBeans development environment to create user-friendly user interfaces that displayed both the content being compressed and the output generated by the compression. A purposive sampling technique was used to select text files with varying complexities from the Google data store, as well as other text documents developed and manipulated by the researcher to meet the level of complexity required for the research experiment. The results showed that each compression algorithm compressed differently in terms of word count. This is due to the fact that the compression ratios of the number of words in each file changed in each compression algorithm. Furthermore, the complexity of the text in the file had no effect on the algorithms used. This was due to the fact that no significant changes in the various algorithms were observed with the selected files, despite the complexities of some files. Finally, time for compression was found to decrease with decreasing file size, though there were some notable variations across all four algorithms used. Before choosing a compression algorithm for efficient text compression, users must first determine their goals.

Introduction

Data compression, also known as source coding, is the process of encoding information using fewer bits (or other information-bearing units) than an unencoded representation would require through the use of particular encoding schemes (Devale, Modi, and Coelho, 2015).This is done in order to reduce the size of the data as much as possible. The ZIP file format is a common example of compression that many computer users are familiar with (Kuhn, Kim, Lopuz, Kuhn, Kim & Lopuz, 2015). The primary goal of data compression is to represent the data in question using as little space as possible (Garca, Ramrez-Gallego, Luengo, Bentez, & Herrera, 2016). This is accomplished by identifying and exploiting data redundancies (Alakuijala, Farruggia, Ferragina, Kliuchnikov, Obryk, Szabadka & Vandevenne, 2018).

Depending on the outcome of the algorithm in question, data compression can be lossy or lossless" (Sun, Ma, Sun & Liu, 2019). Lossy means that some data was lost during the encoding (compression) or decoding (decompression) processes, while lossless means that no data was lost during either process (Wiseman, 2015; Hussain, Al-Fayadh & Radi, 2018). It is critical to note that lossless compression algorithms are used when data loss will not distort the intended information carried by the data to the receiver. However, any loss in the data, even if it is only an alphabet, can change the meaning of the word or sentence, causing the entire information to be distorted. When considering the efficiency and accuracy of the algorithm, the concept of

encoding and decoding is critical (Ma, Hempel, Peng & Sharif, 2012). Huffman, Arithmetic, Run-Length, and Lempel Ziv Welch (LZW) are some of the most commonly used data compression algorithms (Patel, Bhogan & Janson, 2013; Boopathiraja, Kalavathi & Chokkalingam, 2018). The importance of text compression in our time cannot be overstated, given the increase in data generation among all computer users (Xu & Durrett, 2019). Many companies and governments generate terabytes of data every day, and offshore backups are becoming more common for these organisations and countries, as well as with the Cloud Computing new winds blowing strongly in industry. Not to mention user data accessibility options, which are expanding and, as a result, require fast processing, transmission, and storage to meet the needs of modern technology users (Demirkan & Delen, 2013; Tikkinen-Piri, Rohunen & Markkula, 2018).

Despite the fact that the complexities of processing power and storage capacities are increasing in accordance with John Moore's law, the rate of data generation far outpaces the growth of hardware and transmission complexities. This necessitates the attention of field professionals in order to meet the growing user and industry requirements.

The Windows operating system is widely used in homes and businesses, particularly with the Windows 7 release being the industry defector (Egan, 1996). Many text compression algorithms have been tried on this operating system, either embedded or through the use of third-party software installed on the system (Ravi, Raghunathan, Kocher & Hattangady, 2004).

This article investigated, via a comparative study, the aforementioned four (4) major data compression algorithms to determine which is the most efficient and also fast in both encoding and decoding without loss of the input text with future demands in mind. Data compression is very important because it helps to reduce the use of expensive resources such as disc space and transmission bandwidth (Wang, Feng, Chen, George, Bala, Pillai, & Satyanarayanan, 2018). Furthermore, compressed data takes far less time to transmit than uncompressed data (Shanmugasundaram & Lourdusamy, 2011). Despite the fact that compressed data has numerous advantages, decompressing such data necessitates the use of special applications and equipment, which incur additional costs (Wu, Tan & Xiong, 2016). Data compression normally results in a certain amount of data being discarded, the amount of distortion introduced when using lossy compression techniques, and the high-level resources required to compress and decompress the specific data (Azar, ayeh, Makhoul & Couturier, 2022). The efficiency of a compression technique thus includes both the compression and decompression expenses, such as the resource requirement, as well as the degree of compression, the amount of distortion introduced when using a lossy compression technique, and the high-level resources required to compress and uncompress the data (Zou, Yu, Tang & Chen, 2014; Alsheikh, Lin, Niyato & Tan, 2016; Jayasankar, Thirumal & Ponnurangam, 2021).

The main goal of this thesis is to determine which of the many algorithms will best compress text data by eliminating most of the trade-offs during the compression and decompression stages while maintaining a good compression ratio. As a result, the thesis compares which of the text compression Algorithms named best compress given text data in terms of compression ratio, compression speed, and output size within the Windows 7 operating system.

Research Question

1. How different are the compression size of text compression algorithms?

2. What is the effect of the number of words in a text or file size to the compression ratios of text compression algorithms?
3. Does the complexity of a text file affect the compression ratio of text compression algorithms?
4. Does the compression time depend on the file size?

Compression Algorithms

Run-Length Coding

It has been observed that data normally contains series of similar bytes of information. These similar occurrences can be replaced with a set of bytes in order to reduce the size of the data. The act of replacing the similar occurrences with a byte sequence is the idea of Run-length coding. A unique pointer *N* is needed in the data that does not occur as part of the data stream itself. This N-byte can also be recognised if for example all 256 possible bytes can occur in the data stream by using byte stuffing. To demonstrate this, we can describe the exclamation mark (!) to be the N-byte found in the data. One occurrence of an exclamation mark is deduced as the byte during decoding. Two repeated exclamation marks are represented as one exclamation mark occurring within the data. "The method can be described as follows: if a given byte occurs at least four (4) times in a row, then the number of occurrences is counted as one. The compressed data contain the byte, followed by the N-byte and the number of occurrences of the byte. Remembering that we are compressing at least four consecutive bytes, the number of occurrences can be offset by –4. This allows the compression of between four and 258 bytes into only three bytes. For instance the character "***A***" appears four times in a row and can be encoded as *A!:* The uncompressed data look like this; ***ABAAAAWANAOMI*** whiles the compressed data using the Run-Length technique becomes; ***ABA!NAOMI***. It can be seen that the algorithm has been able to reduce the original thirteen (13) characters which is equivalent to thirteen bytes to only nine (9) characters or bytes. When such sequences are found in a given text of data, it is convenient to compress such data by using the Run-Length technique. An example of such data includes icon, line drawings, animations etc. One disadvantage of this compression is that in extreme cases, the output files turn to be larger than the input file which the defiles the aim of data compression.

Huffman Coding

This method is named after D. A. Huffman, who developed the procedure in the1950s. Given the characters that must be encoded, together with their probabilities of occurrence, the Huffman coding algorithm determines the optimal coding using the minimum number of bits. Huffman code procedure is based on the two observations. More frequently occurred symbols will have shorter code words than symbol that occur less frequently. The Huffman code is designed by merging the lowest probable symbols and this process is repeated until only two probabilities of two compound symbols are left and thus a code tree is generated and Huffman codes are obtained from labelling of the code tree. (Sashikala, 2013).

According to Blelloch (2013), the algorithm is now probably the most prevalently used component of compression algorithms, used as the back end of GZIP, JPEG and many other utilities. The most frequently occurring characters are assigned to the shortest code words. A Huffman code can be determined by successively constructing a binary tree, whereby the leaves represent the characters that are to be encoded. Every node contains the relative probability of

occurrence of the characters belonging to the sub tree beneath the node. The edges are labelled with the bits 0 and 1. A Huffman tree is a special binary tree called a tree which is a tree in which a 0 represents a left branch and a 1 represents a right branch. The numbers on the nodes of the binary tree represent the total frequency, F, of the tree below. The leaves of the tree represent the elements, e, to be encoded. The elements are assigned the encoding which corresponds to their place in the binary tree.

Arithmetic coding

Arithmetic coding (AC) is a form of entropy encoding used in lossless data compression. Normally, a string of characters is represented using a fixed number of bits per character, as in the ASCII code. When a string is converted to arithmetic encoding, frequently used characters will be stored with fewer bits and not-so-frequently occurring characters will be stored with more bits, resulting in fewer bits used in total.

Arithmetic coding algorithm encodes an entire file as a sequence of symbols into a single decimal number. The input symbols are processed one at each iteration. The initial interval [0, 1) (or [0, 1]) is successively divided into subintervals on each iteration according to the probability distribution. The subinterval that corresponds to the input symbol is selected for next iteration. The interval derived at the end of this division process is used to decide the "code word" for the entire sequence of symbols.

Lempel Ziv Welch Coding (LZW)

This compression algorithm was developed by Abraham Lempel, Jakob Ziv, and Terry Welch. LZW is a 'dictionary-based' lossless compression algorithm that scans a file for data patterns that appear more than once. These patterns are then saved in a dictionary, and references are placed within the compressed file wherever repetitive data occurs.

The LZW Algorithm is divided into two parts: the encoding algorithm, which converts strings to integer codes, and the decoding algorithm, which does the opposite. Both encoder and decoder algorithms have a default table or dataset that serves as the initial model for both encoder and decoder. As the algorithm runs, new integer codes for various string patterns are added to this table. The number of table entries in the code table is usually set to 4096. When encoding begins, the code table only contains the first 256 entries, with the remaining entries being blanks. LZW detects repeated sequences in the data and adds them to the code table as the encoding progresses. Each code from the compressed file is decoded through the code table to determine which character or characters it represents.

Methodology

The current study was carried out using a quantitative research methodology. Positivists are the most common term used to describe researchers who employ quantitative tools, methods that place an emphasis on counting and measuring. Positivists hold that there is only one truth, or one external reality, and that reality is fixed, directly measurable, and knowable. The researcher chose this strategy because the goal of the study is to determine which of the four (4) main data compression algorithms is most effective and quick in both encoding and decoding without losing the input text.

Sampling

According to Wikipedia (2022), sampling is all about carefully selecting an aspect of the total population to find out their characteristics which gives complete idea about the overall entities. Purposive sampling, also known as judgmental, selective or subjective sampling, is a type of non-probability sampling technique whereby the researcher uses his/her own discretions to choose the unit to be investigated.

Lund Research Ltd (2012) explains that purposive sampling only pays attention to the traits of the population which are of immense interest to him. This eventually will help the researcher to answer the already posed research questions.

The researcher based on the above, selected text files based on their size and complexity. These files were selected from the goggle data store with various complexities and other text documents developed and manipulated by the researcher to meet the kind of complexity needed for the research experiment. The selected sample files can be found in table 1.

Table 1- *Selected Sample Files for the Research*

Input File Name	File size	Number of Words
CONQUEST	768712	11224

psychounix	683952	14218
drugs	595096	5015
Googlebooks-free-all	406344	6268
2400adfaq	339776	7968
ATOMIC	260376	2740
goldenbaton	254360	2103
Manual	203080	4522
CBSOLVE	193104	4146
221bakerst	151984	4238
221baker	148416	4245
2400ad	104304	2397
AQUEST	69344	1199
institute	45728	1209
ampasswords	17080	817

The researcher created a user interface to surface the manipulation of the files into bits, which is often not friendly to interact with, for simple interactivity. The researcher created the interface depicted in Figure 1 using the NetBeans development environment's Graphical User Interface (GUI) designer.

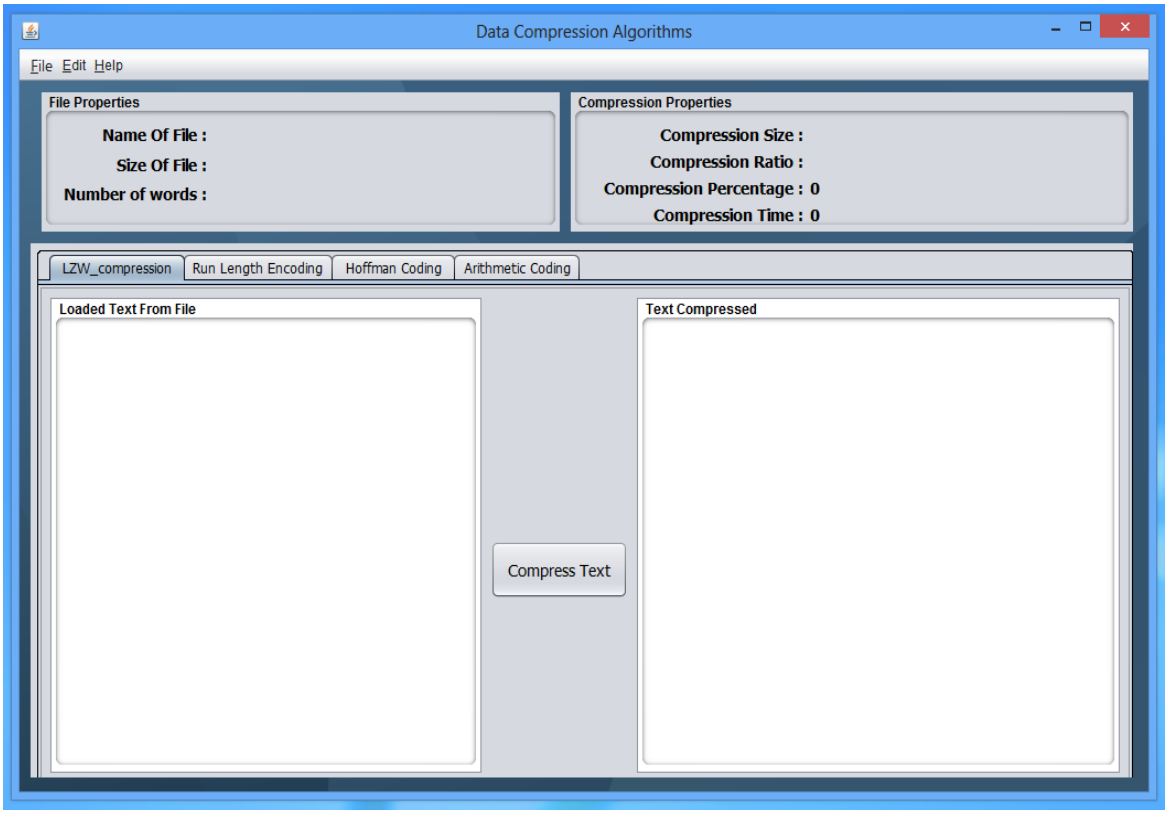

Figure 1: *Designed Interface*

The user loads the text file to be compressed through a menu on the interface, and there is an exit button to end the process. Saving time on the experiment, files are loaded once and are then loaded into all of the algorithms' file text boxes. The directory path of the loaded text, the file size, and the number of words are all displayed in the files properties area for each file that has been loaded into the text box. These are not included in the compressed time.
Each tab is made to command a specific compression algorithm and have a compressed button. These algorithms are timed throughout, from beginning to end. The number of bits compressed and the compression ratio, both of which are not included in the calculated time for the compression, are recorded and displayed along with the time difference.
Once the files have been loaded, all of the algorithms are executed, and their recordings are taken before the next file is loaded. As a result, the experiment's time spent reading text files is cut down.

Figure 2 displays an example of a human compression used in the study.

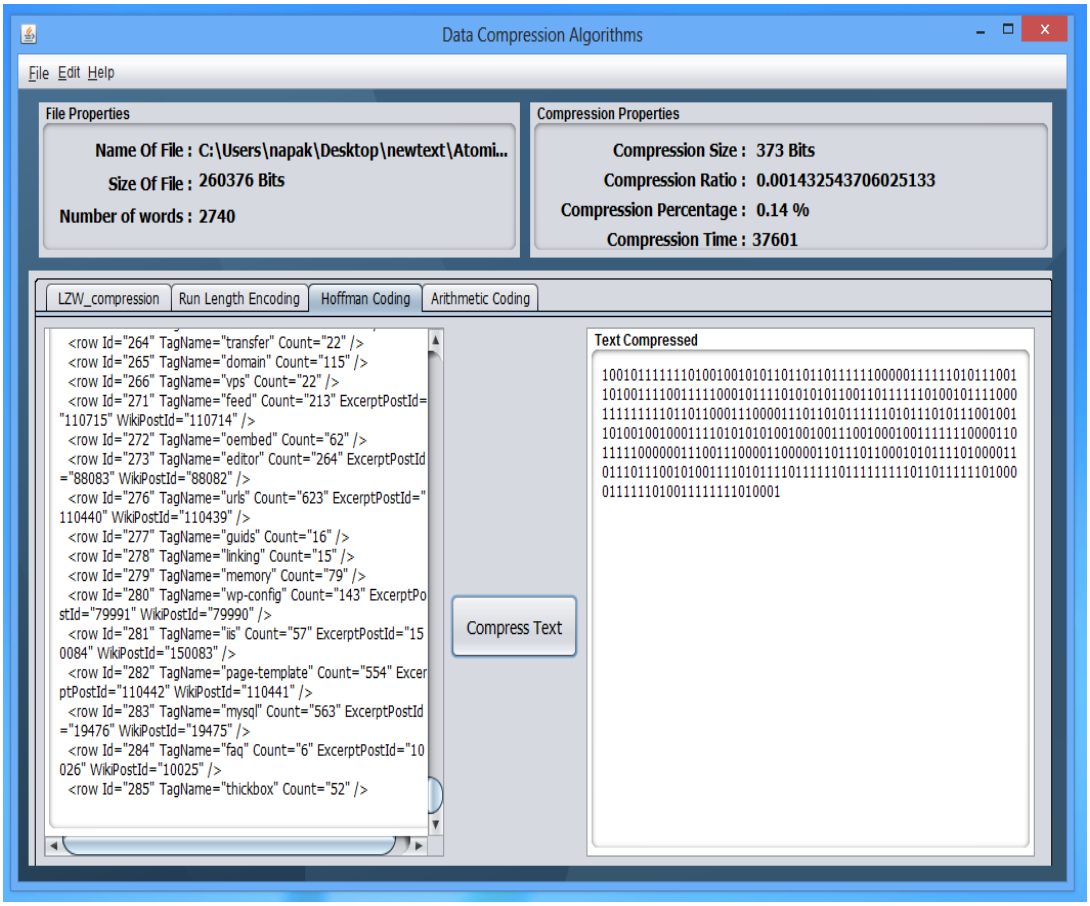

Figure 2: *Samples of Compressed file.*

In order to analyse the data, the researcher used a comparative method in which the chosen sample files were contrasted with one another in light of the study's goals. To ascertain whether one variable is related to another, analyses were carried out. The study also looked at association and causality, with association being the degree to which one variable is dependent on another. Then, the measure of association aids in understanding the connection between the variables. For this study, a correlation was used, which is a single number that indicates how closely two variables are related.

The researcher ran three simulations on each sample text used, and the mean values were then recorded and used to make the variables in the research stable and thus reliable. Additionally, this assisted in averaging out the variations in computer computation and processing speed. On the described computer specification, the values used in the study could therefore be regarded as being extremely stable. Also the researcher immediately after the simulation was run recorded the data to ensure the reliability of the findings. The information was gathered from primary and

secondary sources. The experimental results served as the primary source, and the literature served as the secondary source.

Text files of various sizes and compositions that were downloaded from the internet as well as the outcomes of the simulation served as the main sources of data for the study. The files were found through various internet searches at various sources. A few of the text files that were used combined a number of different text files found online. In order to provide the level of complexity required for the research simulation, this was done to assist the researcher.

The simulation that was run using the listed text data from the internet search produced the primary data that was used in the analysis. These data were trustworthy because they were produced by the computer using simulation and were free from interference from other function formulas. Regarding the timing of the loop and the execution of the involved algorithms, there were no human interruptions during the simulation.

A sizable amount of publicly available data on data compression and its related academic topics was used as the secondary source of data. To give the work a little more polish, the bases and availability of the data were confirmed. A certain amount of information and experimental findings were also accessible from university academic resources, including academic books and, in particular, academic journals from online databases in the field.

Using the chosen sample files, the experiments' simulations of the four compression algorithms produced their results. Microsoft Excel was then used to tabulate and record the results. One of the Microsoft Office suite's tools, Microsoft Excel, is utilized for mathematical computations. Excel also gives users the option to create various graphs from the data they have stored. Tables and graphs were then used to analyse the data.

Results and Discussion

The research questions were tested against the four compression algorithms to find the possible answers to them. Each was tested by running the various algorithms to find their reactions. Observations were made and conclusions drawn at the end. The questions are as follows:

Research Question One: How different are the compression size of text compression algorithms?

This research question was analysed by using grouped bar graph to determine how different are the compression size of text compression algorithms. The results is shown in Figure 3

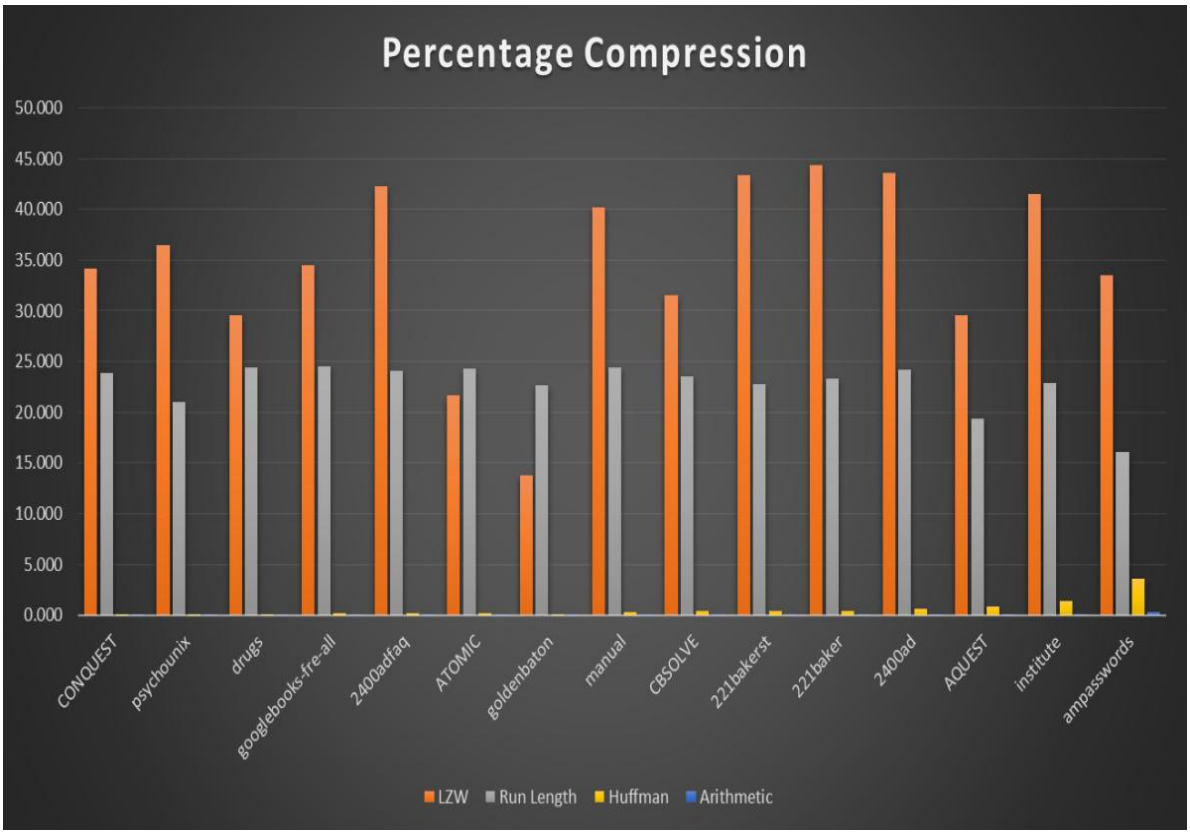

Figure 3: Percentage Compression Difference of selected Files

The figure shows that the Lempel Ziv Welch coding algorithm had the greatest compression difference on all of the selected files except "Atomic" and "goldenbaton," for which the Run Length coding algorithm had the greatest compression difference. Run-Length Encoding algorithm had the second highest compression ratios, however overtook LZW algorithm in two of the selected files mainly "Atomic" and "goldenbaton". In terms of compression difference between the selected files, the Huffman algorithm came in third place. Although the difference was not statistically significant, it did result in a significant increase in compression difference with the last two least selected files. The bar graph shows that the smallest file in terms of size, "ampassword," had the greatest compression difference with the Huffman algorithm. Finally, when compared to the other three compression algorithms, the Arithmetic algorithm recorded the smallest compression difference in all fifteen (15) selected files.

Research Question Two: What is the effect of file size on compression ratios of text compression algorithms? A bar graph was used to determine the effect of file size on compression ratios of text compression algorithms. The results is shown in Figure 4

Input File	N of Words	LZW	Run Length	Huffman	Arithmetic
CONQUEST	11224	34.202	23.817	0.104	0.008
psychounix	14218	36.490	20.995	0.137	0.008
drugs	5015	29.572	24.417	0.126	0.009
googlebooks-fre-all	6268	34.494	24.472	0.203	0.013
2400adfaq	7916	42.222	24.032	0.214	0.017
ATOMIC	2740	21.657	24.343	0.143	0.022
goldenbaton	2103	82.198	48.030	0.112	0.022
manual	4522	40.212	24.428	0.349	0.026
CBSOLVE	4146	31.586	23.497	0.373	0.033
221bakerst	4238	43.403	22.761	0.403	0.041
221baker	4245	44.377	23.354	0.413	0.038
2400ad	2397	43.576	24.229	0.671	0.058
AQUEST	1199	29.524	19.337	0.836	0.075
institute	1209	41.460	22.879	1.441	0.129
ampasswords	817	33.525	16.101	3.536	0.345

Figure 4: The effect of number of words on compression ratios of selected files

Figure 4 shows a bar graph with the results of the four algorithms and their compression ratios for the selected files. The Huffman and Arithmetic algorithms both increase compression ratios as the number of words in the file decreases with little inconsistency. Arithmetic coding had the lowest compression ratio of 0.008 for the largest file, while Huffman had the highest at 1.104. As the size of the individual files decreased, the ratios for both algorithms increased. They all recorded a maximum ratio for ampassword, which has the fewest files of all the selected files. Arithmetic had a higher ratio of 0.345, whereas Huffman had an equivalence of 3.536.

In contrast, the LZW and Run-Length algorithms recorded erratic file size ratios. LZW recorded the highest ratio of 82.2 for goldenbaton, which falls in the middle of the file size spectrum. Ironically, Atomic had the lowest ratio of 21.66, which is just above goldenbaton in terms of file size. The rest of the ratios recorded by the files were erratic because no pattern was followed. In addition, Run-Length also recorded the highest compression ratio of 48.03 for the same file "goldenbaton" as recorded by LZW algorithm. It however recorded the least ratio of 16.10 for the least file "ampassword" which happens to be the least file in terms of file size. The rest of the ratios recorded by the files were less than 25 and inconsistence as far as their file sizes are concern.

From the above figures it can be seen that the number of words or the file size of a given file has not got much effect on the compression ratio of that file.

Research Question Three: Does the complexity of a text file affect the compression ratios of text compression algorithms? To determine the complexity of a text file affect the compression ratios of text compression algorithms a bar was used as shown in figure 5 below:

Input File	File size	LZW	Run Length	Huffman	Arithmetic
CONQUEST	768712	34.202	23.817	0.104	0.008
psychounix	683952	36.490	20.995	0.137	0.008
drugs	595096	29.572	24.417	0.126	0.009
googlebooks-fre-all	406344	34.494	24.472	0.203	0.013
2400adfaq	339776	42.222	24.032	0.214	0.017
ATOMIC	260376	21.657	24.343	0.143	0.022
goldenbaton	254360	13.765	22.678	0.112	0.022
manual	203080	40.212	24.428	0.349	0.026
CBSOLVE	193104	31.586	23.497	0.373	0.033
221bakerst	151984	43.403	22.761	0.403	0.041
221baker	148416	44.377	23.354	0.413	0.038
2400ad	104304	43.576	24.229	0.671	0.058
AQUEST	69344	29.524	19.337	0.836	0.075
institute	45728	41.460	22.879	1.441	0.129
ampasswords	17080	33.525	16.101	3.536	0.345

Figure 5: Compression ratio Difference of Selected Files

Figure 5 depicts the selected compression files, their sizes, and the compression difference when run through the compression algorithms. The files that are considered complex are shaded. That is, they contain xml and other symbols that are not found in ordinary text. With the exception of LWZ, the other coding schemes reduced the compression difference from the largest to the smallest file with minimal variations. Variations in the drop were observed from "drugs" to "institute," which, while reduced downwards, was not consistent. In contrast, the Huffman Coding scheme increased consistently from the largest file to the smallest recording, 3.536 and 0.104, respectively. Arithmetic Coding Scheme also followed Huffman's lead, increasing from 0.008 to 0.345 from the largest to the smallest file.

Research Question Four: ***Does the compression time depend on the text complexity?*** A simple bar graph was used to show whether the compression time depend on the test complexity as shown in Figure 6 below:

Input File	File size	LZW	Run Length	Huffmann	Arithmetic
CONQUEST	768712	110625	20	249755	229991
psychounix	683952	84941	5	193043	160503
drugs	595096	73312	8	132596	115125
googlebooks-fre-all	406344	36246	10	62175	57930
2400adfaq	339776	40066	6	50825	50076
ATOMIC	260376	19710	10	33870	30010
goldenbaton	254360	53191	2	25986	24700
manual	203080	13531	1	19910	17973
CBSOLVE	193104	13741	2	2165	2180
221bakerst	151984	12921	6	12846	11014
221baker	148416	10853	4	11788	10401
2400ad	104304	7558	3	7473	6988
AQUEST	69344	2358	1	1712	1722
institute	45728	207	1	1018	966
ampasswords	17080	889	1	187	250

Figure 6: Compression Time for the Selected Files.

The time used to compress each of the selected files by the algorithms was compared with the sizes of the files. Figure 6 illustrates the comparisons of these calculated durations. From Figure 6, it is observed that the compression time generally decreases with decreasing files size but with some exceptions. Arithmetic, LWZ, and Huffman coding pulled larger values than Run Length Encoding. Run Length was not consistent although its maximum value twenty (20) was for the file with the largest file and three of the last three least file all recorded a maximum time of one (1) millisecond.

It can be seen from the Figure 6 that "CBSOLVE" dropped out of pattern as its time for all the other three algorithms were far lesser than "221barkerst" except LZW algorithm in which "CBSOLVE" had a higher time of 13741 milliseconds than "221barkerst" with a total time of 12921 milliseconds. In addition, in the LZW algorithm, it can be seen that there were some inconsistencies in the time used after the first three largest files. "2400adfaq" with file size 339776 had much time 40066 milliseconds than "googlebooks-free-all" with larger file size of 406344 which used a total time of 36246 milliseconds.

Furthermore, “goldenbaton” with file size 254360 used much time of 53191 milliseconds than Atomic with larger file size of 260376 which used a maximum time of 19710 milliseconds.

Run-Length Algorithm also experienced some time inconsistencies apart from “Conquest” which had a largest time of 20 milliseconds with the largest file size of 768712. The rest from psychounix to CBSOLVE experienced a toggle of time between 1 millisecond for manual and 10 milliseconds for both googlebooks-free-all and Atomic. Among the most complex files which are conquest, drugs, googlebooks-free-all, atomic and goldenbaton, even the time was inconsistence. For example, apart from conquest which had a maximum time of 20 milliseconds, googlebooks-free-all and Atomic with 10 milliseconds, and drugs with 8 milliseconds, goldenbaton which is also recorded a compression time of 2 milliseconds which is far lesser than 2400adfaq and 221barkerst which recorded a maximum of 6 milliseconds although not considered as a complex file.

In conclusion, it was generally observed that compression time largely depends on the complexities of the input file. Conquest which is one of the complex files recorded the highest time in all the four used algorithms in the research. Conquest recorded the highest time of 110625 milliseconds in LZW, 20 milliseconds in Run-Length, 249755 milliseconds in Huffman and finally recorded 229991 milliseconds in Arithmetic algorithm.

Practical Application of data compression

Data compression is very significant because it helps to curtail the use of exorbitant resources like disk space and transmission bandwidth. Data compression helps in the reductions in storage hardware, data transmission time, and communication bandwidth. This can result in significant cost savings. Compressed files require significantly less storage capacity than uncompressed files, meaning a significant decrease in expenses for storage. A compressed file also requires less time for transfer while consuming less network bandwidth. This can also help with costs, and also increases productivity.

Conclusion and recommendation

In general, each compression algorithm compressed differently based on the number of words. This is due to the fact that the compression ratios of the number of words in each file changed in each compression algorithm. Furthermore, the complexity of the text in the file had no effect on the algorithms used. This was due to the fact that no significant changes in the various algorithms were observed with the selected files, despite the complexities of some files. Finally, time for compression was found to decrease with decreasing file size, though there were some notable variations across all four algorithms used. Before choosing a compression algorithm for efficient text compression, users must first determine their goals. They must decide whether they want to compress the file as quickly as possible without regard for the outcome of the compressed file. If this is the case, they should think about using the Run-Length Encoding algorithm. Furthermore, the choice of a compression algorithm must be based on the complexity of the file to be compressed. Finally, the researcher suggests that in future research of this type, more files of varying complexity and larger file sizes be used to make the results and findings more holistic.

Limitations of the study

This article compared the compression ratio, compression time, compression size and file complexity of selected files using Huffman, Arithmetic, Run-Length, and Lempel Ziv Welch coding algorithms within the Windows 7 Operating System. Since the simulation and the experiments were conducted in Windows 7 Operating System, there is the possibility that the results will not be the same in other operating systems.
In addition, different sample files may also behave differently in the four compression Algorithms used, since they may differ in terms of file size, file complexity, and data types. When audio and video files are used in this experiment, it may yield different outcomes. It has put to the public domain the differences that exist between the four compression algorithms in respect to compression time, compression ratio, and how file with diverse complexities react to selected compression algorithms.

References

Alakuijala, J., Farruggia, A., Ferragina, P., Kliuchnikov, E., Obryk, R., Szabadka, Z., & Vandevenne, L. (2018). Brotli: A general-purpose data compressor. *ACM Transactions on Information Systems (TOIS)*, *37*(1), 1-30.

Alsheikh, M. A., Lin, S., Niyato, D., & Tan, H. P. (2016). Rate-distortion balanced data compression for wireless sensor networks. *IEEE Sensors Journal*, *16*(12), 5072-5083.

Azar, J., Tayeh, G. B., Makhoul, A., & Couturier, R. (2022). Efficient Lossy Compression for IoT Using SZ and Reconstruction with 1D U-Net. *Mobile Networks and Applications*, *27*(3), 984-996.

Boopathiraja, S., Kalavathi, P., & Chokkalingam, S. (2018). A hybrid lossless encoding method for compressing multispectral images using LZW and arithmetic coding. *Int J Comput Sci Eng*, *6*, 313-318.

Demirkan, H., & Delen, D. (2013). Leveraging the capabilities of service-oriented decision support systems: Putting analytics and big data in cloud. *Decision Support Systems*, *55*(1), 412-421.

Egan, E. A. (1996). The Era of Microsoft? Technological Innovation, Network Externalities, and the Seattle Factor in the US Software Industry.

García, S., Ramírez-Gallego, S., Luengo, J., Benítez, J. M., & Herrera, F. (2016). Big data preprocessing: methods and prospects. *Big Data Analytics*, *1*(1), 1-22.

Hussain, A. J., Al-Fayadh, A., & Radi, N. (2018). Image compression techniques: A survey in lossless and lossy algorithms. *Neurocomputing*, *300*, 44-69.

Jayasankar, U., Thirumal, V., & Ponnurangam, D. (2021). A survey on data compression techniques: From the perspective of data quality, coding schemes, data type and applications. *Journal of King Saud University-Computer and Information Sciences*, *33*(2), 119-140.

Kuhn, D., Kim, C., Lopuz, B., Kuhn, D., Kim, C., & Lopuz, B. (2015). Chapter 6: Archiving and Compressing Files. *Linux and Solaris Recipes for Oracle DBAs*, 139-155.

Ma, T., Hempel, M., Peng, D., & Sharif, H. (2012). A survey of energy-efficient compression and communication techniques for multimedia in resource constrained systems. *IEEE Communications Surveys & Tutorials*, *15*(3), 963-972.\

Patel, D., Bhogan, V., & Janson, A. (2013). Simulation and comparison of various lossless data compression techniques based on compression ratio and processing delay. *International Journal of Computer Applications*, *81*(14).

Ravi, S., Raghunathan, A., Kocher, P., & Hattangady, S. (2004). Security in embedded systems: Design challenges. *ACM Transactions on Embedded Computing Systems (TECS)*, *3*(3), 461-491

Savant, M., Devale, J., Modi, U., & Coelho, S. (2015). Data Solution. The International Journal of Science and Technoledge, 3(4), 68.

Sayood, K. (2018). Introduction to Data Compression, (Fifth Edition) A volume in The Morgan Kaufmann Series in Multimedia Information and Systems Book. Elsevier Inc.

Shanmugasundaram, S., & Lourdusamy, R. (2011). A comparative study of text compression algorithms. *International Journal of Wisdom Based Computing*, *1*(3), 68-76.

Sun, X., Ma, H., Sun, Y., & Liu, M. (2019). A novel point cloud compression algorithm based on clustering. *IEEE Robotics and Automation Letters*, *4*(2), 2132-2139.

Tikkinen-Piri, C., Rohunen, A., & Markkula, J. (2018). EU General Data Protection Regulation: Changes and implications for personal data collecting companies. *Computer Law & Security Review*, *34*(1), 134-153.

Wang, J., Feng, Z., Chen, Z., George, S., Bala, M., Pillai, P., ... & Satyanarayanan, M. (2018, October). Bandwidth-efficient live video analytics for drones via edge computing. In *2018 IEEE/ACM Symposium on Edge Computing (SEC)* (pp. 159-173). IEEE.

Wiseman, Y. (2015). The still image lossy compression standard-JPEG. In *Encyclopedia of Information Science and Technology, Third Edition* (pp. 295-305). IGI Global.

Wu, M., Tan, L., & Xiong, N. (2016). Data prediction, compression, and recovery in clustered wireless sensor networks for environmental monitoring applications. *Information Sciences*, *329*, 800-818.

Xu, J., & Durrett, G. (2019). Neural extractive text summarization with syntactic compression. *arXiv preprint arXiv:1902.00863*.

Zou, H., Yu, Y., Tang, W., & Chen, H. W. M. (2014). FlexAnalytics: a flexible data analytics framework for big data applications with I/O performance improvement. *Big Data*

Research, *1*, 4-13.