Optimising Storage Resource using Morpheme based Text Compression Technique

Rockson Kwasi Afriyie
Department of Computer Science
Kwame Nkrumah University of Science and Technology, Ghana

J. B. Hayfron-acquah
Department of Computer Science Kwame Nkrumah University of Scienceand Technology, Ghana

Joseph K. Panford
Dpt. of Computer Science
KNUST, Ghana

ABSTRACT
In this paper, we present a text compression technique which utilises morpheme-based text compression to optimise storage resources. The proposed technique is designed to decompose words into their morphemes and then to produce code representations for compression. The proposed algorithm is implemented using English Language text data and applied using 30 different texts of different lengths collected from different sources with different natures. The efficiency increases with the increase in the number of long, repetitive morphemes in the input data. To the best of our knowledge, the resulting implementation is the first to demonstrate lossless compression using such a technique. We illustrate its suitability and effectiveness on a number of benchmark file sizes – small, middle-sized, large, and very large real-world application. The results indicated a good compression performance of 98% making the approach an attractive one. A further virtue of this method is its dynamic application. A degraded compression can be compensated for by appending identified morphemes within the document to the dictionary to improve compression. The evaluation experiments show that: if storage space is the primary consideration, the morpheme-based text compression technique is an efficient approach for compressing text data.

General Terms
Text compression: Morpheme

Keywords
Algorithm, morpheme, clean data, storage resource

1. INTRODUCTION
The demand and dependence on digital data and information keep on increasing. New knowledge, ideas and technologies are created and added daily into the annals of organisations, institutions, governments, and even individuals; their daily digital data and document files needs keep on growing to significant sizes. It is asserted that "The web is creating massive amounts of data every second of the day [15]. Moore's law states that the amount of digital information increases tenfold every five years and the Social Web is accelerating data" [15]. A special report in The Economist, Andrew Brust [2] observed that we are at the point of an "industrial revolution of data," with vast amounts of digital information being created, stored and analysed. The rise of "big data" has led in turn to an increased demand for tools to both visualise and analyse the information. The New Competition Is Data; Data Complications" [15]. The information in schools, offices, libraries, books, etc is mostly textual data which keeps on increasing daily. Projects such Gutenberg, Distributed Proofreaders and Digital Text Community are converting already published books into eBooks [5, 17]. They are electronically preserving the literary material and other media of the world for everyone to use [5]. Digital Text Community is digitising "ink-on-paper" text such as books, periodicals, documents [8] and most hitherto physical libraries are also being digitised. The arrival of total digitised education or world would surely be the beginning of data storage trauma. This will be the time and period when every learnable and learnt material in schools, colleges and universities can only be accessed electronically.

1.1 Theory of Data Compression
The theoretical background of compression is provided by information theory. Information theory is a study of information based on probability theory [7]. It was proposed by Claude E. Shannon at Bell Laboratories in 1948 [12] and based on people's reception and reaction towards given information; which is aimed at a mathematical way of measuring the quantity of information [7]. Data compression is the fundamental expression of information theory which is a branch of mathematics concerned with various questions about information, including different ways of storing and communicating messages [12]. Data compression is connected to the field of information theory because of its concern with redundancy [12]. Redundancy is the number of bits used to transmit a message minus the number of bits of actual information in the message. According to Wikipedia, redundancy refers to "the amount of wasted "space" used to transmit certain data" [17]. Data compression is a way to reduce or eliminate unwanted redundancy. Therefore, the basis of any data compression algorithm depends on the presence of redundancy in the given data. The more redundancy the source data has, the more effective a compression algorithm may be [7].

1.2 Early Development in Data Compression
According to history of irreversible data compression, the idea of creating sounds by adding together pure tones goes back to antiquity [18]. Following work by Joseph Fourier around 1810, it became clear by the mid-1800s how smooth function could be decomposed into sums of sine waves with frequencies corresponding to successive integers. Early telephony and sound recording in the late 1800s already used the idea of compressing sounds by dropping high-and low-frequency components [18]. Early days of television in the

1950s saw similar kinds of compression for images but serious efforts were made when digital storage and processing of images became common in the late 1980s [18]. Morse code used in telegraphy was a form of earliest text compression [18]. Modern data compression began in 1940s with the development of information theory [18]. In 1949, Claude Shannon and Robert Fano introduced the method of compressing information using probabilities of blocks and in late 1970s, software compression programs based on adaptive Huffman coding were developed [18]. In 1977, Abraham Lempel and Jacob Ziv published a paper with a different approach to the symbol coding problem; their method is the basis of most lossless data compression techniques [11, 18]. The late 1980s and early 1990s saw digital images compression standards and lossy compression methods respectively [18]..

1.3 Data Compression Algorithms

A data compression algorithm refers to the step-by-step methodologies employed to compress data into a relatively small size with the aim of converting source data at the compression end into a compressed message [7] and restore it precisely the same content information or with little distortion. Compression algorithms can be divided into two major families [12]. They are lossless compression and lossy compression [11, 13]. The lossless compression which is applied to database records, spreadsheets, or word processing files, encodes the original input without loss of information. The lossy compression accepts a slight loss of data and proves effective when applied to graphics, images and digitised voice [12]. An algorithm may be static or adaptive Figure 1 and Figure 2 respectively base on its ability to adjust itself. The compression algorithm which samples from the inputs to adjust the model is called adaptive compression [13]. A fixed model compression algorithm is called static modelling compression.

1.4 Characteristics of Existing Compression Algorithms

There are two major ways to compress data, Statistical-based model and Dictionary-based model [6]. The choice of either method is informed by the type of data being compressed. The statistical-based algorithms explore the numerical redundancy of individual characters within the data.

1.5 Statistical-Based Compression Method

The statistical-based algorithms explore the numerical redundancy of individual characters within the data. It is based on counting individual symbol and the determination of its frequency of occurrence. This technique achieves compression by assigning shorter codes to most frequently used symbols and longer codes to rare appeared symbols [14]. Among the compression techniques that use statistical criterion are Huffman coding algorithm, Shannon-Fano algorithm, run-length coding and arithmetic algorithms [7].

1.6 Dictionary-Based Data Compression

A dictionary-based compression scheme uses a different concept [12]. A coding system (coder) keeps a pattern of characters in memory called dictionary, which keeps string patterns seen before and the indices are used to encode the repeated patterns [7]. It reads in input data and looks for groups of symbols that appear in a dictionary. If a string match is found, a pointer or index into the dictionary is output instead of the code for the symbol. When the match is longer, the compression ratio is better, however, the size of the dictionary may affect the efficiency of the algorithm. In general, dictionary-based compression replaces phrases (symbols) with pointers. If the number of bits in the pointer is less than the number of bits in the phrase, compression will occur. However, the methods for building and maintaining a dictionary are varied. Some popular dictionary-based algorithms are LZ77 and LZ78 compression algorithms.

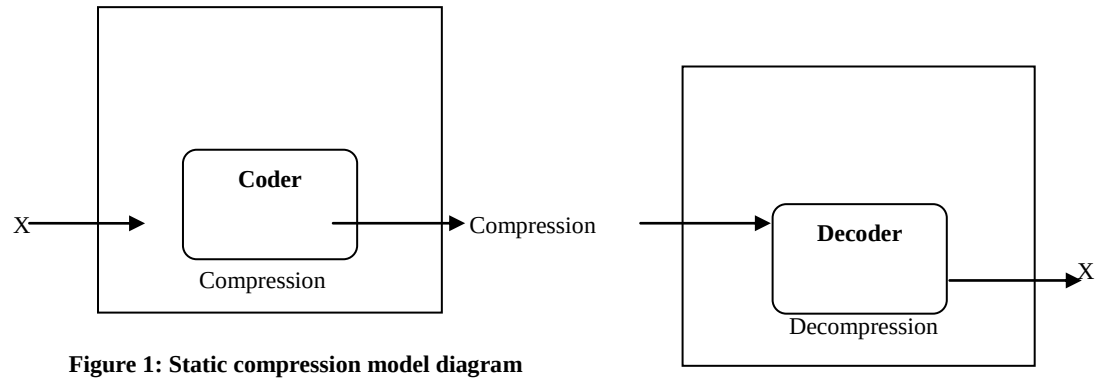

Figure 1: Static compression model diagram

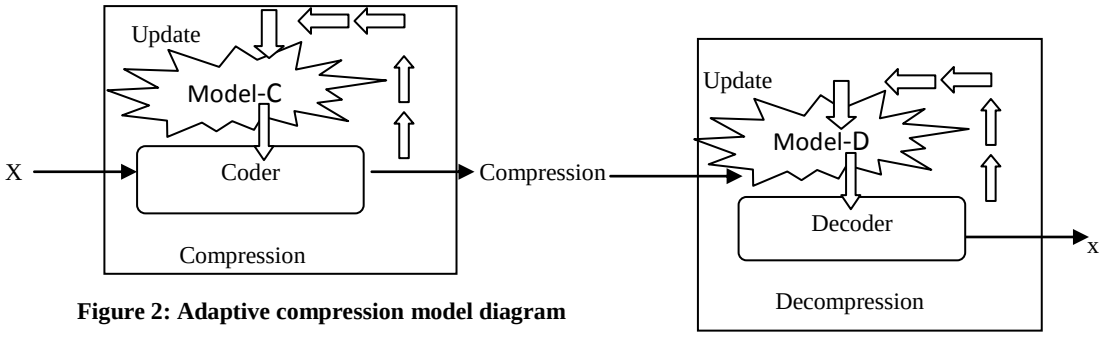

Figure 2: Adaptive compression model diagram

1.7 Recent Development in Data Compression

Data compression as an area for research work is generally not new. According to Ibrahim Akman and his colleagues, most text compression algorithms perform compression at character level or at word level [1]. Jan Lansky and Michal Zemlicka reported that there are two basic types of text compression by symbol representation – the characters and words. They believe that, the syllable is another way by which text can be compressed [9]. They claimed that syllable, character and word compressions will be suitable for middle, small and very large files respectively. Today's advances in lossless file compression, generally assume that there is an inherent redundancy in text and therefore seek to minimise or eliminate these redundancies [7]. Most text compression algorithms perform data compression at character level or word level and they do not consider adjacent string structures in words such as syllable [1]. The existing text compression techniques have been reported to have some weaknesses [1]. At the character level, a single bit error may be sufficient to result in a long stream of errors in the coded file, at the syllable level, the compression ratio does not meet the desired storage requirements and the most effective compression algorithms are reported to be computationally expensive [1].

1.8 Overview of Morpheme Concept

A morpheme is the smallest bit of language that has its own meaning; either a word or a part of a word [3]. Every language has a morpheme and every word comprises one or more morphemes. They can be classified in different ways. A morpheme may be free or bound [10]. Free morphemes can stand alone and function independently as words; while bound morphemes appear only together with other morphemes or as parts of words, always in conjunction with a root. Bound morphemes can be split further into derivational or inflectional [5, 10]. Derivational morphemes are the type, whose part of speech is modified upon root combination. For instance, in the word *joyfulness*, the addition of the bound morpheme *-ness* to the root *joyful* changes the word from an adjective (*joyful*) to a noun (*joyfulness)*. Inflectional morphemes cause a change to a noun's number or verb's tense leaving the word's meaning or class intact. They are achieved by addition of *–s or –ed* to the root. For example, adding *-s* to the root *dog* to form *dogs* and adding *-ed* to *wait* to form *waited*. English has eight inflectional affixes – affixes that depend on the function of a word in a sentence. For example, the inflectional affix's' on the end of *pot* makes the word plural. The remaining affixes in English are derivational affixes, which change the form or meaning of words. Table 1 lists all eight of the inflectional affixes in English.

Table 1: The inflectional affixes in English Language

Inflectional Affixes	Derivational Affixes			
S: creates plural nouns	Noun	Verbs	Adjective	Adverb
S: creates possessive nouns	-ant	-ate	-able	-ly
S: creates third person singular verbs	-er	-en	-al	-ward
Ed: creates verbs past tense	-hood	-ise	-ful	-wise
en: creates past participle	-ment	others	-y	others
ing: creates present participle	-ness		-ous	
er: creates comparatives	-tion		others	
est: creates superlatives	others			

Table 2 labels the various morphemes in a typical English sentence: *"The students have a wonderful teacher"*

Table 2 : Illustration of morphemes

The	stud ent	s	hav e	a	won der	ful	teac h	er
free	free	bound	free	free	free	boun d	free	boun d

2. MATERIALS AND METHODS

2.1 Research Design

Experimental research design was used to obtain information about the population by selecting and experimenting sample from the entire population. Scientific sampling procedure was used to select 30 text data files from different sources of different natures for the implementation and experimentation of the proposed algorithm.

2.2 The Proposed Algorithm

The proposed morpheme-based algorithm works using English text data as follows:

Miss – "Mis" and "s"; Rejoyce – "Re", "joy" and "ce"; displays – "dis", "play" and "s" etc.

The proposed algorithm is as follows:

i. Read the text input

ii. Partition the string input into possible morphemes

iii. Extract morphemes.

iv. Scanned the extracted morpheme through the dictionary to do a match if there exist.

v. Represent the morphemes with the relevant code representation

vi. If the string in the input text is not in the morpheme unit passed text unaltered.

The operation of the algorithm is further illustrated in Figure 3 using a flow chart.

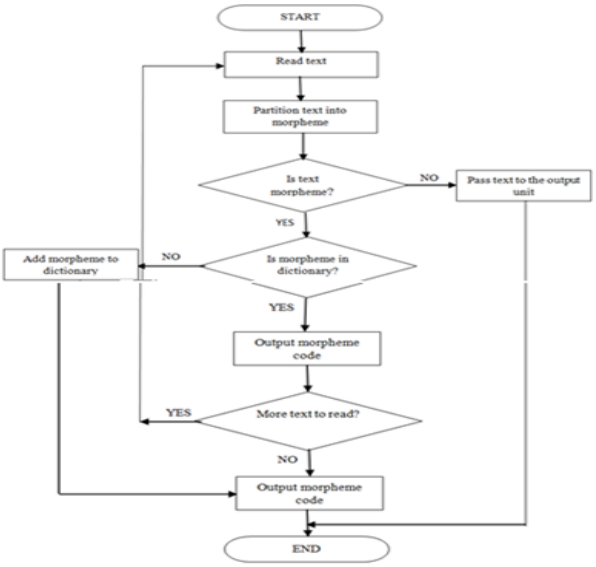

Figure 3: Morpheme-based text compression algorithm flowchart

Table 3: Alphabets Number Assignment

Char	Code
Blank	0
A	1
B	2
C	3
D	4
E	5
F	6
G	7
H	8
I	9
J	10
K	11
L	12
M	13
N	14
O	15
P	16
Q	17
R	18
S	19
T	20
U	21
V	22
W	23
X	24
Y	25
Z	26

Alphabets number assignment shown in Table 3 was used as the basis for coding to generate the morpheme dictionary. This assignment scheme made the generation of the dictionary simple and computationally inexpensive.

2.3 MORPHEME DICTIONARY GENERATION

2.3.1 Morpheme coding system

The morphemes were coded by a simple approach of adding the coded numbers for each character. The target morpheme assigned, for instance, the morpheme "ing" was coded as follows:

Step 1: assign the constituent characters in the morpheme with their respective numbers

i = 9

n = 14

g = 7

Step 2: sum them up

9 + 14 + 7 = 30

From the calculation performed, the morpheme "ing" would be assigned the correspondent code representation 30. Thus, in the dictionary, the morpheme "ing" would be represented by the code 30. All the other morphemes were coded likewise and were assigned code representations. Using the same process of coding, the morpheme dictionary was obtained as shown in Table 4.

2.4 Demonstration of the Proposed Algorithm for English Language Text Data

The English language is used for the implementation of the proposed algorithm. The choice of English Language was its widespread; global language; most important and popular language worldwide [4]. Again, English is considered the most widely and commonest language on the Internet [16].

English sentence is selected for the illustration of the proposed approach since this is the language used in implementation.

"The display of joyfulness by miss joyce in playing computer games using a joystick is enjoyed by her friends".

The given sentence contains 19 words but the resultant morpheme count is 27.

Algorithm

i. Read the sentence word by word

ii. Send each word to the morphemes unit one by one to be partitioned into its morphemes. For example, morphemes unit partitions the string "display" into two morphemes:

display as "dis" and "play".

joyfulness as "joy" "ful" and "ness"

miss as "mis" and "s"

joyce as "joy" and "ce"

playing as "play" and "ing"

computer as "compute" and "r"

games as "game" and "s"

using as "us" and "ing"

joystick as "joy" and "stick"

enjoyed as "en", "joy" and "ed"

her as "he" and "r"

friends as "friend" and "

2.5 Theoretical Consideration of the Proposed Algorithm

We took the sentence under consideration morpheme-by-morpheme vis-a-vis the number of times each morpheme is repeated in the given sentence. For example, "joy" reoccurred four (4) times. This proposed algorithm proposes that, if the morpheme "joy" is encoded and represented by a token (code representation) which is less than the number of characters involved instead of the usual conventional requirements of this individual character being repeated, there could be some significant saving of storage space of a storage resource. The detailed breakdown of the sentence and the inherent character redundancies is done below:

s ⟶ 3-1 = 2 redundant characters

joy ⟶ 4-1 = 3 redundant characters

play ⟶ 2-1 = 1 redundant character

ing ⟶ 2-1 = 1 redundant character

r ⟶ 2- 1 = 1 redundant character

The morpheme-based technique represents the original 27 morphemes in only 19. Certainly, the actual format used for the storage of text is generally binary rather than ASCII morphemes like this, but the principle remains the same.

2.6 Compression Performance Metrics

According to Ida Mengyi Pu, "the compression effect of an algorithm is measured by the amount of shrinkage of the source file in relation to the size of the compressed version" [7]. From this assertion, the following formulae are provided:

$$CR = Lac / Lbc \quad (1)$$

where CR represents compression ratio; Lac represents the size after compression and Lbc size before compression.

$$CR = Lac / Lbc$$

$$CR = 19 / 27$$

$$Cf = Lbc / Lac \quad (2)$$

$$= 27 / 19$$

where Cf represents compression factor

$$Sp = [(Lbc - Lac)/Lbc] * 100 \quad (3)$$

$$= [(27-19)/27] * 100$$

$$= 29.6\% \text{ (1 dp.)}$$

where Sp represents saving percentage

$$Cg = 100 - (Lac/Lbc) * 100 \quad (4)$$

$$= 100 - (19/27) * 100$$

$$= 29.6 \text{ (1 dp.)}$$

where Cg represents compression gain.

2.7 Decompression Component

The decompression method is the same as the compression aspect. It uses the same components as the compression algorithm. The compressed text whose first string is a word is read since the morpheme is not modified during decompression. Coded text data is replaced by the original morpheme in the word. At the decompression, the string replaces the coded representation. The code will be the same as the one created since during compression the order of morpheme remains same during compression/decompression process. At the same time, the alphabetic text is transmitted to the output unit. If the entry text is a coded representation, then its code is searched for and replaced. The searching and replacement process is continued till the end of the file is reached and finally, the original text is produced from the compressed text.

2.8 Implementation of the Proposed Algorithm

The performance of the algorithm was measured using 30 text files of sizes ranging from 10 Kilobytes to 7,181 Kilobytes. The texts were selected from different sources. The data collected for the experiment were grouped into four categories: namely, small-sized files (file_size < 100 kilobytes), middle-sized files (file_size <200 kilobytes), large-sized files (file_size <1000 kilobytes), and very large-sized (file_size >1000 kilobytes). The summary of the categories is indicated in Table 5.

2.9 Experimental Design and Testing Environment

Hewlett-Packard Pentium IV of system type 32-bit operating system; processor of type Intel(R) Atom(TM) CPU N455 of speed 1.66 GHz machine with an installed memory (RAM) of 1.00 GB, a Hard disk capacity of 217 gigabytes and having Microsoft Windows XP as its Operating System was used to run the various texts.

3. RESULTS AND DISCUSSIONS

3.1 Data Classification

The data collected for the experiment were grouped into four categories: namely, small-sized files (file_size < 100 kilobytes), middle-sized files (file_size <200 kilobytes), large-sized files (file_size <1000 kilobytes), and very large-sized (file_size > 1000 kilobytes).

3.2 Small-Sized File Category

The ten (10) different files range from 10KB to 81 KB. The experimental results in Table 6 showed gain of storage space increased from 3.9% to 96.5%. On the average, there was an overall saving percentage of 59.39% in the small-sized category.

The outcome of the experiment is summarised using graph in Figure 4.

Table 4: Morpheme dictionary

Morpheme	Index	Morpheme	Index	Morpheme	Index
a	1	ster	62	ings	49
arch	30	er	23	ies	33
be	7	vice	39	semi	46
un	35	let	37	ingly	67
in	23	ie	14	fully	76
non	43	hood	42	ily	46
dis	32	ship	52	est	44
de	9	dom	32	es	24
inter	66	ry	43	e	5
mis	41	ing	30	en	19
mal	26	ful	39	s	19
pseudo	80	ite	34	ers	42
super	79	an	15	er	23
out	56	ist	48	ence	27
sur	58	ism	41	ences	46
sub	42	or	33	ably	40
over	60	rs	37	ement	57
under	62	ant	35	ements	76
hyper	72	ment	52	ely	42
ultra	72	al	13	ations	78
mini	45	age	13	ation	59
co	18	ness	57	ance	23
pan	31	ise	33	ances	42
anti	44	en	19	ally	50
pro	49	less	55	ors	52
trans	72	ly	37	ments	71
fore	44	like	37	ively	73
pre	39	y	25	ive	36
post	70	ish	36	ity	54
ex	29	ian	24	ions	57
re	23	ic	12	ion	38
uni	44	ive	36	ward	46
mono	57	ous	55	ed	9
bi	11	able	20	wise	56
di	13	wards	65	ure	44
multi	75	proto	84	ry	43
tri	47	poly	68	counter	96
neo	34	auto	57		

Table 5: Text categories

SSF (file_size<100KB	MSF (file_size<200KB)	LSF (file_size<1000KB)	VLSF (file_size>1000)
10	101	349	1,169
11	104	438	1,177
14	128	567	2,109
18	144	621	2,123
18	152	800	7,181
37	152		
38	175		
43	182		
44	183		
81	199		

MSF:Middle-sized File SSF:Small-sized File

VLSF: Very Large-sized File LSF:Large-sized File

Table 6: Experimental result and compression gain of small-sized file category

File #	Category	Original size (KB)	Compressed size (KB)	Saving percentage
SSF 1	Researcher mock text 1	10	0.1	89.1
SSF 2	Curriculum	18	4.6	70.4
SSF 3	A paper on data revising dictionary	81	68.8	15.1
SSF 4	Bible text	11	0.4	96.5
SSF 5	Dr. Gordon Moore IC 1965 article	18	15.9	11.9
SSF 6	Data compression literature	44	42.3	3.9
SSF 7	Article on polytechnic education	37	8.1	78.1
SSF 8	Readme "Calgary News"	14	2.4	83.0
SSF 9	Educational committee report	43	5.4	87.5
SSF10	Project documentation Software Eng.	38	17.2	54.8

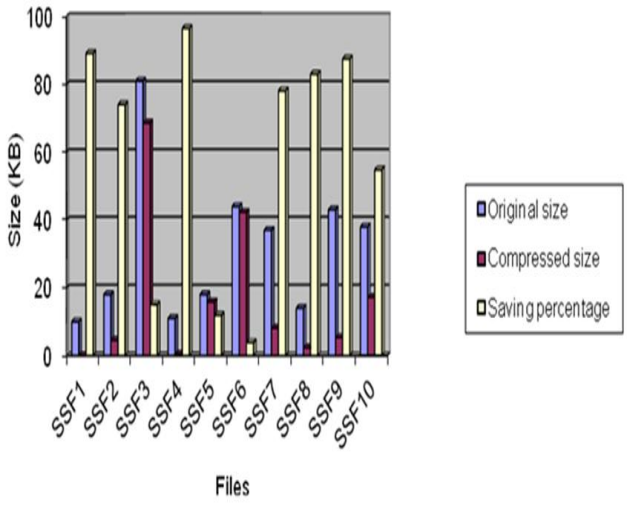

Figure 4: Compression gain for small-side files

3.3 Middle-sized Files Category

The middle-sized file (MSF) category consists of collection of text data of sizes smaller than 200 kilobytes, (file_size <200 kilobytes). The ten (10) different files in this category ranged from 101KB to 199 KB. The experimental results are summarised in Table 7. The results show gains in storage resource space from 3.5% to 83.3%. The average saving percentage is 34.53%. The results are further graphed as shown in Figure 5.

3.4 Large-sized Files (LSF) Category

The experimental results of this category show saving percentages from 44.3% to 77.4% as summarised in Table 8 with average overall percentage of 63.54%. The results are further indicated in Figure 6.

3.5 Very Large-sized Files Category

A close inspection of Table 9 suggests that better compression percentages were recorded for larger texts for the compression technique. It made gains from 18.5% to 96.4%. In general, an average of 60.62% compression percentage was obtained for files whose size is larger than 1000 Kilobytes and it gradually increases as the file size gets larger as indicated in Table 9. Figure 7 establishes the graphical aspect of the very large-sized file category.

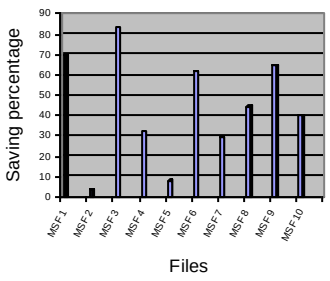

Figure 5: Compression percentage for middle-sized files

Table 7: Experimental results of middle-sized file category

File #	Category	Original size (KB)	Compressed size (KB)	Saving percentage
MSF 1	Student Project work	104	31.5	69.7
MSF 2	E-book "Double Your Dating"	175	168.8	3.5
MSF 3	Text collection on algorithm	144	24.1	83.3
MSF 4	Mini project in Software Engineering	101	68.5	32.1
MSF 5	Bank of Ghana 2005 annual report	183	168.1	8.2
MSF 6	Book MS. Word program	199	76.4	61.6
MSF 7	MSC Thesis	152	107.5	29.3
MSF 8	Oxford journal paper"	128	71.0	44.5
MSF 9	Skills assessment report	152	54.0	64.5
MSF10	Article on mobile telephone	182	109.0	40.1

Table 8: Experimental Results of Large-sized File Category

File #	Category	Original size (KB)	Compressed size (KB)	Saving percentage
LSF 1	Researcher mock text 2	567	231.4	59.2
LSF 2	UEW Thesis	621	233.2	64.4
LSF 3	Book "Getting Started with ICT	800	445.3	44.3
LSF 4	project "Using Rhythmic Pattern to improve Pupil's handwriting	349	79.0	77.4
LSF 5	Thesis "Adoption of e-banking in Ghana"	438	111.9	74.4

Table 9: Very Large-sized Files Category

File #	Category	Original size (KB)	Compressed size (KB)	Saving percentage
VLSF 1	Researcher mock text 2	2,123	76.42	96.4
VLSF 2	UEW Thesis	1,117	682.4	42.4
VLSF 3	Book "Getting Started with ICT	1,169	953.9	18.5
VLSF 4	project "Using Rhythmic Pattern to improve Pupil's handwriting	7,181	112.2	98.4.4
VLSF 5	Thesis "Adoption of e-banking in Ghana"	2,109	1,099.0	47.8

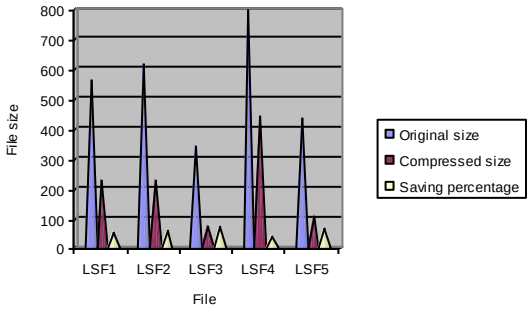

Figure 6: Compression gain in large-sized file category

Figure 7: Compression gain in very large-sized file category

3.6 Comparison of Compression Percentages for the Four Categories of File Sizes

From Figure 8, the compression percentages for the four categories of files fluctuated. SSF category recorded a highest compression gain of 96.5% and a lowest compression gain of 3.9%. MSF category had a highest compression gain of 83.3% and also depreciated to a lowest percentage of 3.5%. Again, LSF recorded a highest compression gain of 77.4% and a lowest compression gain of 44.3%; while VLSF category had a highest gain of a storage resource space of 98.4% with a least gain in storage space of 18.5%. It is important to note that saving percentages for the four categories follow similar trends depending only on the size of the source files as shown in Figure 8. This means that the content of the text does not affect the performance of the proposed morpheme-based compression technique.

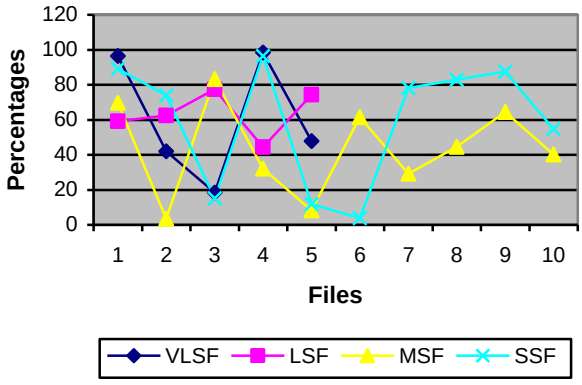

Figure 8: Compression percentages for the four categories of file sizes

4. CONCLUSION

Considering the analysis, it could be inferred that on the average, as the size of data increases, the performance of the proposed algorithm also improved. This could be observed among the very large-sized file (VLSF) category. VLSF 3 of size 1,169KB had a saving percentage of 18.5% and VLSF 1,177 KB recorded 42.0%. Equally, VLSF 1 of size 2,123KB recorded 96.4% compression gain, while VLSF 4 of size 7,181KB obtained a compression percentage of 98.4%. These observations could confirm the report of IDC in 2011 that most of the digital content is not unique. They reported that nearly 75% of our digital world is a copy; only 25% is unique. Again, it was noted that saving percentages for the four categories follow similar trends depending only on the size of the source files as shown in Figure 8. This means that the content of the text does not affect the performance of the proposed morpheme-based compression technique. It was also observed that morpheme-based text compression technique was very suitable for all manner of file sizes – small, middle, large, or very large. This could be so because the proposed morpheme-based compression took advantage of the character, word and syllable methods of data compression.

5. ACKNOWLEDGMENTS

Authors are thankful to Dr. M. Asante, Head of Department, Computer science, Kwame Nkrumah University of Science and Technology (KNUST), Kumasi.

6. REFERENCES

[1] Akman, I. et'al. 2011. Lossless text compression technique using syllable based morphology. The International Arab Journal of Information Technology, Vol. 8, No. 1, January 2011.

[2] Andrew, B. 2011. Big Data. www.zdnet.com/blog/service...of...data...growing.../4750 http://www.zdnet.com/five-big-data-trends-revolutionizing-retail-7000019510/

[3] International Dictionary of English. 2002. Morpheme is the smallest bit of language. Low Price Edition, University Press.

[4] David G. 2000. The future of English; A guide to forecasting the popularity of English Language in the 21st century.

[5] Distributed proofreaders. 2011. Digitisation of Public Domain Books. http://www.pgdp.netg. (accessed 2011 October 20).

[6] http://www.codeguru.com/cpp/cpp/algorithms/compression/article.php/c5089/ (accessed 2010 August 13).

[7] Ida, M. P. 2006. Fundamental data compression. Butterworth-Heinemann Linacre House, Jordan Hill, Oxford OX2 30 Corporate Drives, Suite 400, Burlington, MA 01803.

[8] Jon, N. 2007. Digital Text Community — new forum on digitizing "ink-on-paper" texts. http://www.teleread.com /ebooks/digital-text-community-new-forum-on digitizing-ink-on paper-texts/ (accessed 2011 October 24).

[9] Lansky, J., Zemlicka, M. 2005. Text Compression: Syllables Conference: Databases, Texts, Specifications, Objects - DATESO, pp. 32-45, 2005 http://academic.research.microsoft.com/Publication/1873500/text-compression-syllables. (Access 2011 December 14).

[10] Mark, C. 2003. An Introduction to Language. ENG 346: Aspects of the English Language Lesson 4: Morphology.Updated January 7, 2003. http://www.uncp.edu/home/canada/work/markport/language/aspects/spg2003/04morph.htm. (accessed 2012 April 3).

[11] Mark, N. 1989. Data compression, LZW Data Compression http://marknelson.us/1989/10/01/lzw-data-compression.

[12] Mark, N. and Jean-loup, G. 1995. The Data compression book 2nd edition, M&T Books, Wiley, New York, NY. http://staff.uob.edu.bh/files/781231507_files/The-Data-Compression-Book-2nd-edition.pdf

[13] Shenfeng, C. 1996. Algorithmic Applications of data Compression Techniques. Department of Computer Science; Duke University. http://www.cs.duke.edu/~reif/paper/chen/chen.thesis/chen.thesis.pdf.

[14] Skibi´nski, P. Grabowski, S. Z., and Deorowicz, S. 2005. Revisiting dictionary-based compression Software Practice and Experience. (accessed 2012 January 11).

[15] Social Media Informer. 2012. Data is growing and shows no signs of slowing down. http://www.socialmedia

informer.com/data/information/statistics/ (accessed 2012 December 4).

[16] Web Technology Survey. Usage of content languages for websites.http://w3techs.com/technologies/overview/content_language/all

[17] Wikipedia. 2012. Converting already published books into eBooks. en.wikipedia.org/wiki/Gutenberg (accessed 2012 May 5).

[18] Wolfram, S. 2002. A new kind of science. Notes for Chapter 10: Processes of Perception and Analysis Section: Data Compression Page 1069. http://www.wolfram_science.com/nksonline/page-1069b-text?firstview=1 (accessed 2011 March 10).