View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by Covenant University Repository

Lossless Text Compression Technique Using Syllable Based Morphology

Ibrahim Akman[1], Hakan Bayindir[1], Serkan Ozleme[2], Zehra Akin[3], and Sanjay Misra[1]
1Computer Engineering Department, Atilim University, Turkey
2Parana Vision Image Processing Technologies and Solutions Consultancy Corporation, Turkey
3Meteksan Systems and Computer Technologies Corporation, Turkey

Abstract: *In this paper, we present a new lossless text compression technique which utilizes syllable-based morphology of multi-syllabic languages. The proposed algorithm is designed to partition words into its syllables and then to produce their shorter bit representations for compression. The method has six main components namely source file, filtering unit, syllable unit, compression unit, dictionary file and target file. The number of bits in coding syllables depends on the number of entries in the dictionary file. The proposed algorithm is implemented and tested using 20 different texts of different lengths collected from different fields. The results indicated a compression of up to 43%.*

Keywords: *Algorithm, text compression technique, syllable, multi-syllabic languages.*

Received December 15, 2008; accepted August 3, 2010

1. Introduction

The Data Compression (DC) is not only the cost effective technique due to its small size for data storage but it also increases the data transfer rate in data communication. A data compression algorithm should emphasize the originality of the data during compression and decompression process. This property is called lossless compression. Today, available lossless text compression techniques are generally based on the assumption that a text contains a large amount of redundancy and each of these techniques addresses to different types of redundancies. Most text compression algorithms perform compression at character level or at word level [11, 22, 24] and they do not consider adjacent string structures in words such as syllables which may provide important advantages [4, 14]. The existing text compression techniques have a number of other weaknesses. First, in many cases, a single bit error is sufficient to result in a long stream of errors in the coded file. Second, the compression ratio of the existing utilities is not as large as desired for storage applications [4]. Lastly, the most effective compression algorithms are reported to be computationally expensive as also pointed by [13].

Text compression based on syllables is a relatively new area of research (see for example, [2, 16, 20, 25]). The syllable-based text compression may be very useful especially for languages with rich morphologies (e.g., Turkish, Czech and German). In these languages, the words usually consist of several syllables and syllables play the role of natural transition between letters and words [16]. Syllables are usually longer than one character and each word contains at least one syllable [5, 17]. In many cases, different words contain the same syllables in their structures. As the syllables are somewhere between characters and words, it can be expected that syllable compression could take advantage of both character compression and word compression. Further, the HTML pages are normally smaller (in size) and syllable-based compression may be the most appropriate technique for their transfer in networking and communication since the syllable based compression is reported to be the most appropriate approach for small documents [16]. Actually, syllable based compression has recently been studied by Lansky and his colleagues [17, 20, 23]. These studies utilize databases of frequently used syllables and mainly adapted well-known algorithms of adaptive Huffman coding and LZW to use syllables and words instead of characters. They reported in [17] that the results are ambiguous for Czech. They also left open the applicability of syllable-based compression for different languages [17, 19].

Against this backdrop, we propose to take syllabic nature of multi-syllabic languages into account for text compression. The proposed approach is different than previous syllable based compression approaches [16, 20, 25] in that, it uses syllables as the basic unit and compresses these fragments utilizing an automaton to produce a volatile dictionary and the approach is based on an original modular lossless compression algorithm. The remainder of this paper is organized as follows. A brief review of classification of languages according to their syllabic nature and the proposed syllable based text compression technique are given in section 2. An example based on the proposed algorithm is demonstrated in section 3. The decompression technique is discussed in section 4. Theoretical considerations and implementation are outlined in sections 5 and 6. The discussions and conclusions drawn constitute the last two sections.

2. Classification of Languages and the Proposed Syllable Algorithm

A possible classification of languages [9] considering their syllabic nature is as follows:

- Mono-syllabic Languages: These are formed from words of one syllable only. Chinese and Japanese are examples of such languages.
- Multi-syllabic Languages: These are formed from words of one or more syllables. This category has two sub classes as follows:

a. Languages which are formed by adding affixes to their roots. Addition of these affixes may change the root. The Semitic (e.g., Arabic, and Hebrew), Germanic (e.g., English, German and Danish) and Romance (e.g., French, Italian and Spanish) languages are examples of the languages falling into this category.

b. Languages which are formed by adding suffixes to roots or other suffixes. This addition normally does not change the root. These languages are called agglutinative. Turkish and Ural-Altaic languages (e.g., Hungarian and Finnish) are examples of this category.

The roots, prefixes and suffixes appear in the form of syllables in these languages and a model for generalization of their structure is proposed as follows:

$$word = syllable_1 + syllable_2 + ... + syllable_n$$

The main components of the proposed approach are:

- Source file: contains the original text.
- Filtering unit: mainly searches the text for characters not included in the alphabet.
- Syllables unit: divides words into its syllables.
- Compression unit: creates a dictionary for compressing the input text and producing the target file.
- Dictionary: contains different syllables contained in the text and their corresponding binary codes. This file is volatile.
- Target file: contains compressed data.

Of these components, source and target files constitute input and output files respectively. The filtering unit is the first module to process words to search for characters not included in the alphabet (non-alphabetical characters). If such a word is detected then it is partitioned into three segments which are: the string preceding the non-alphabetical character, the character itself and the string following the character. The preceding and following strings are considered as separate words in later stages. The character (or string of such characters) is treated as non-dividable syllable. For this purpose, the filtering unit inserts tags and writes a “no” for all non-dividable strings and “yes” for all dividable strings for all words. These tags are messages received by the syllables unit. The filtering system also detects blanks and punctuation marks and merges them with the last syllable of the divided word later in the syllable unit.

The syllables unit uses a finite automaton to partition filtered words into their syllables. These syllables are then sent to the compression unit without any changes being made to the syllable order. The compression unit processes syllables to create a dictionary and the target file. The dictionary is used as a volatile lookup table during compression and is empty when the compression starts. When a different syllable is entered into the compression unit it is directly written into the target file and, consecutively, its bit representation is created in the dictionary. When the same syllable later arrives at the compression unit its code is found in the dictionary and this code is inserted into the target file. With this approach, the target file is always a combination of plain text and bit representations, and the dictionary is embedded in the target file. Therefore, the dictionary is discarded when the process finishes. This saves memory space and is one of the important advantages of the proposed algorithm. The compression unit uses two flags to distinguish plain text for different syllables and bit representations of repeated syllables in the target file. These flags are needed for decompression. The first flag should actually be a character whose possibility of occurrence is zero in the text (e.g., we used δ in our examples). The second flag which is used at the end of a bit representation is a bit representation of the shortest possible length and is composed of ones only (e.g., 11, 111 or 1111). Finally, the proposed algorithm is case sensitive.

3. Demonstration of the Proposed Algorithm: a Worked Example

A Turkish sentence is selected to illustrate the proposed approach since this is the language used in implementation. The example sentence is: “Alexander heranda hersey olacak der ve Alexis ise olanın aleme anlatılmamasını ilave eder“ (means “Alexander says anything will happen at any time and Alexis adds that this fact should not be told to the world”). The given sentence contains 13 words and is read by the filtering unit word by word. Each time a word is read, filtering unit seeks for words containing a letter (or a string of letters) which is not included in the alphabet. In our example, the first word entered is “Alexander” as shown in Table 1. The letter “x” is not included in Turkish alphabet and therefore, “Alexander” is partitioned into “Ale”, “x” and “ander” with tags “Y”, “N” and “Y” respectively as shown in Table 1. The same procedure applies to “Alexis”. All the other words remain as they are and take the tag “Y”. In this example, “λ” stands for the blank between words and is used to make it easier to follow for the reader. Each word is then sent to the syllables unit one by one to be broken down into its syllables. For example, syllables unit partitions the fourth string “heranda” into three syllables as “her”, “an” and “daλ” as shown in Table 2. This unit extracts 39 syllables.

Extracted syllables are received by the compression unit one after the other. Of these syllables, 22 are different. Therefore, only 22 of the syllables are coded in the dictionary. For example, “an” is a syllable and first detected in the word “A-le-x-an-der”. It is repeated 3 times in the text (“her-an-da” and “an-la-tıl-ma-ma-sı-nı”). With its first occurrence, the binary representation is determined as “100” in the dictionary unit and other occurrences are not considered. However, “veλ” is not repeated and represented by “1101” as shown in Table 2. The syllables are written as they are into the target file when they are observed for the first time. For example, the first word is partitioned into its syllables as “A-le-x-an-derλ” and these syllables are written into the target file as they are as shown in Table 2. In the next set of syllables the syllable**s** “her” and “daλ” directly go to the dictionary file in which their codes are created and target file since they appear for the first time. However, the middle syllable “an” was detected in “A-le-x-an-der” and its binary representation “100” was created then. Therefore, “herδ-100-111-daλ” is written into the target file. The flag “δ” indicates that binary code representation will follow the plain text and possible shortest string of binary digit “1” (i.e., “111”) is used to indicate that binary code representation for a syllable has been completed and plain text will follow.

4. Decompression

The decompression algorithm is the same as its compression counterpart. The decompression algorithm uses the same main components as that of compression algorithm and can be summarized as follows:

- Data is read from the file (whose first entry is a word since new syllables are not modified during compression) and then a check is applied for the flag of data. The input mode is changed according to the type of flag.
- The inputted data is sent to filtering unit if it is an alphabetical string. The filtering unit searches for a character not included in the alphabet and inserts tags. This stage is bypassed for binary codes since they are aliases for syllables that have been encountered before. The inputted word is then sent to syllables unit and is partitioned into its syllables according to its tag. Binary data bypasses this stage since it does not need to be partitioned but to be replaced with original syllable.
- Finally, data arrives at the decompression module. If this data is a word, then it’s directly recorded to dictionary and its binary code is created. This code will be the same as the code created during compression procedure since the order of syllables is not changed during compression and decompression. Concurrently, this alphabetic data is sent to the output module. If the inputted data is a binary representation then its code is searched in the dictionary file since it must have been encountered before. As a result of the search process the matching entry is found and sent to output unit.

The above steps are repeated until the end of the file is reached.

5. Theoretical Considerations

The size (s_o) of the original file is measured in terms of bits and can be given by the following formula:

$$s_o = \sum_{i=0}^{c_c-1} l_e(character_i) \quad (1)$$

where $l_e()$ is a function that gives the encoding length of a given character. The character count (c_c) and encoding length (l_e) for a particular character in the text are two main parameters effecting original file size in this formula.

Table 1. Source file and output of filtering unit.

Text	Tag	Word	Text	Tag	Word
Alexanderλherandaλheyλ	Y	Ale	Alexisλiseλolanınλalemeλ	Y	Ale
	N	x		N	x
	Y	anderλ		Y	is
	Y	herandaλ		Y	iseλ
	Y	herşeyλ		Y	olanınλ
olacakλderλveλ	Y	olacakλ		Y	alemeλ
	Y	derλ	anlatılmamasınıλilaveλeder	Y	anlatılmamasınıλ
	Y	veλ		Y	ilaveλ
				Y	ederλ

Table 2. Output of syllables and compression units.

Syllables				Dictionary						Target File
				Syll.	Code	Syll.	Code	Syll.	Code	
A	o	seλ	ma	A	00	cakλ	1100	nıλ	11000	Alexanderλherδ 100 111
le	la	o	ma	le	01	veλ	1101	e	11001	daλδ110 1111 şeyλolacakλ
x	cakλ	la	sı	x	10	isλ	10000			δ0101 1111 veλδ0000 0001
an	derλ	nınλ	nıλ	an	100	i	10001			0010 1111 isλiseλδ1010 1011
derλ	veλ	a	i	derλ	101	seλ	10010			1111 nınλδ 0000 0001 1111
her	A	le	la	her	110	nın	10011			meλδ 0100 1011 1111 tılmaδ
an	le	meλ	veλ	daλ	1000	meλ	10100			10110 11111 sınıλδ 10001
daλ	x	an	e	şeyλ	1001	tıl	10101			01011 01101 11111 eδ00101
her	isλ	la	derλ	o	1010	ma	10110			
şeyλ	i	tıl		la	1011	sı	10111			

Compression gain (cg) can be defined as the amount of space recovered as a result of compression and can be calculated by

$$cg = 100 - \left(\frac{\textit{compressed file size}}{\textit{original file size}} \times 100 \right) \qquad (2)$$

where original file size and compressed file size should be in same unit (bits, bytes, Mbytes, *etc.)*.

Compressed file consists of two parts, a unicode part (P_U) and a binary part (P_B) according to the proposed algorithm. Then, compressed file size (s_c) is simply

$$S_c = P_U + P_B \qquad (3)$$

where P_U represents the size of unicode part and P_B is the size of the binary part. Compression gain can therefore be reformulated as:

$$cg = 100 - \left(\frac{P_U + P_B}{s_o} \times 100 \right)$$
(4)

The unicode part (P_U) of the compressed file is the part which includes the syllables encountered for the first time. These syllables use 16 bits per character. Therefore, the space used by these syllables is

$$P_U = \sum_{i=0}^{c_s - 1} charCount(syll_i) \times 16 \qquad (5)$$

where c_s is the number of syllables that are encountered for the first time, char count() is a function that gives character count of a syllable and $syll_i$ is i^{th} first encountered syllable.

Binary Part (P_B) of the file is the portion of the file which includes the exchanged syllable codes which represent the detected repetition. This part is written in pure binary form. Therefore, the space used by binary part of the file is

$$P_B = \sum_{i=2}^{l_{BM} - 1} \sum_{j=0}^{C_{Si} - 1} C_{r_i}(Syll_j) \times i \qquad (6)$$

where l_{BM} is the maximum bit length that can hold all syllables encoded, C_{S_i} is the number of different syllables encountered during the bit length i, $c_{ri}(syll_i)$ is a function that returns the repetition count of $syll_j$ and $syll_j$ is the j^{th} encountered syllable during bit length. In this formula, the maximum bit length l_{bm} is a function which represents the number of bits required to encode all syllables encountered. The maximum number of syllables to be encoded using *n* bits in our algorithm is $2^{n-1}+1$ where n=>2 since our algorithm uses zeroes added to header of the syllable codes and these are length aligners rather than discrete identifiers of different syllables. This means that, in our coding scheme 001 and 000001 are pointing to the same entry in the dictionary. Therefore, the average length of syllables $l_{Syll\,Avg}$ is the amount of space that a discrete syllable occupies on file in terms of bits and can be calculated by

$$l_{Syll\ Avg} = \frac{\sum_{i=0}^{C_{Syl}} charCount(Syll_i)}{C_{Syl}} \times l_e \qquad (7)$$

where C_{syl} is the total number of discrete syllables in the file, l_e is the encoding length for the compressed file, char count() is a function that gives character count of $syll_i$. The average compression gain is then

$$cg = 100 - \left(\frac{\sum_{i=0}^{c_s - 1} charCount(syll_i) \times 16 + \sum_{i=2}^{l_{Syll_{Avg}} - 1} \sum_{j=0}^{C_{S_i} - 1} C_{r_i}(Syll_j) \times i}{s_o} \times 100 \right) \quad (8)$$

This leads to the fact that the upper limit for compression cg_u is reached when L_{BM} reaches $_{\text{syll avg.}}$

6. Experimentations and Validation: Implementation of Proposed Algorithm for Turkish Language

The Turkish language is used for the implementation of the proposed algorithm. Turkish is one of the oldest living languages. It is the sixth most widely spoken language in the world [12] and spread over a large geographical area in Europe, Australia and Asia. The available literature provides few studies targeted text compression on Turkish language [1, 6, 7, 10]. The syllable based text compression on Turkish was not examined properly except the work of Ucoluk *et al.* [25], who used a genetic algorithm based on Huffman encoding upon mixed alphabet of characters and syllables. This approach requires extensive Huffman tree constructions.

Java programming language is selected for the implementation since, with Java, (1) the resulting code will be totally cross-platform (in theory), (2) project can be developed faster with the extensive class support of Java, and (3) development environment may be altered during development and there will be no loss of time due to different operating systems. For implementation, we used a modified version of the finite automaton given by [2] and [17].

6.1. An Overview of Turkish

The Turkish alphabet contains 29 letters and excludes the q, x, and w of the English alphabet. The additional letters are ç, ğ, I, s, ö, and ü, whose corresponding upper cases are Ç, Ğ, I, S, Ö and Ü respectively. The upper case of I is İ. A Turkish text uses blank and punctuation characters. Fundamental morphological characteristics are as follows:

- It is mainly suffix based.
- Its words do not contain gender identification.
- Its words are formed by syllables.
- A word/syllable never starts with ğ (or (Ğ)).
- A word contains at least one syllable.
- A syllable may contain one or more letters.
- The letter is always a vowel for one letter syllables.

According to the Turkish language's vocalic harmony, every syllable contains one vowel (v) and the number of letters can be at most four in a syllable [3]. The first two letters in a syllable cannot be consonants (c) and it is not possible to have two consecutive vowels in a syllable [3]. There are seven regular syllable structures in Turkish [2] as follows:

One v	: (v) o (that)
One v and one c	: (vc) at (throw)
One c and one v	: (cv) ye (eat)
One v, one c and, one v	: (vcv) ara (search)
One c, one v and one c	: (cvc) gel (come)
One v and two c	: (vcc) ilk (first)
One c, one v and two c	: (cvcc) sert (Hard)

The other syllable models are irregular and mainly belong to foreign origin. The most common irregular syllables are [2]:

One c, one v and, three c	:(cvccc) kontr (kontr)
Two c and, one v	:(ccv) gri (gray)
Two c, one v and, one c	:(ccvc) tren (train)
Two c, one v and, two c	:(ccvcc) tröst (trust)
Two c, one v and, three c:	(ccvccc) krankl (crankshaft)

6.2. Implementation

The performance of SA is measured using 20 text files ranging from 4.6 to 726.4 Kbytes. The type of texts is given in two categories in Table 3. The first category generally contains texts collected from different sources of different natures and the second category mainly contains translated or original Turkish stories and novels of different sizes.

The proposed algorithm was also compared against two other lossless compression algorithms, namely Adaptive Huffman coding algorithm and bit-oriented Lempel-Ziv-Welch (LZW) [8, 26] Table 4. The results are evaluated in terms of Compression Percentage (CP), CP= (LO-LC)/LO x100, where LO: Length of original text and LC: Length of compressed text.

A close inspection of Table 4 suggests that better compression percentages were obtained for larger texts for SA. This is because the ratio of the same (incompressible) syllables increases for larger files. In general an average of 36.22% compression percentage was obtained for files whose size is larger than 100 Kbytes and it gradually increases as the file size gets larger as shown in as shown in Figure 1.

It is important to note that compression percentages for the two categories follow similar trends depending only on the size of the source files as shown in as shown in Figure 2. This means that the content of the text does not affect the performance of the proposed technique.

It is fairly easy to rank the performance of different compression algorithms. For smaller files (file_size<100 Kbytes), Huffman and LZW performed better than SA. For larger files (100 Kbytes<file_size<750 Kbytes), except C26 and C29, SA yields better results than Huffman. On the average, Huffman performs better than SA for small files (file_size<100Kbytes) at a rate of 9.8% whereas this percentage is 3.6% in favour of SA for larger files (file_size>100 Kbytes). Although LZW produces better performance than SA for larger files the gap between their average compression percentages was reduced. This gap between SA and LZW is 21.3% for smaller files (file_size<100Kbyte) and 10.42% for larger files (100 Kbytes<file_size<750 Kbytes). For even larger files, SA is likely to perform better than Huffman and LZW since these two are stabilized around 30-35% and 45-50% respectively. Compared to Adaptive Huffman and LZW algorithms, SA is more flexible, continually adapting itself to the text. Although Huffman and LZW algorithms can adapt themselves after a change in file characteristics, they need a relatively long time for adaptation [8]. Additionally, LZW is negatively influenced by the learning period of the compression procedure. Furthermore, compared to Huffman and LZW, SA requires a relatively less memory space since it needs only one volatile dictionary.

It is important to note that computational times follow an increasing trend for both of Huffman and LZW algorithms, which means this trend depends mainly on the size of the source files for these algorithms as shown in Table 4. However, it is interesting to observe that the compression time for SA gradually increases up to a certain size of text and then starts to decrease as shown in Figure 3. A plausible explanation for this observation is that syllables start to repeat themselves after a certain size in which case dictionary file reaches to saturation a point and new codes are rarely needed. This, of course, reduces compression time. This means computation time for SA does not only depend on the size of the file but also depends on its content.

Table 3. Text categories.

File #	**Category-I**	**Size (Bytes) (source file)**	**File #**	**Category-II**	**Size (Bytes) (source file)**
C_{11}	Conference paper on success of MS students	4666	C_{21}	A paper on Oguz Khan	39040
C_{12}	A text explaining a software	13112	C_{22}	A chapter for data structure	57431
C_{13}	A document on the Informatics departments in Turkey	14146	C_{23}	Epic story of Oguz Khan	77570
C_{14}	Software project	17799	C_{24}	A story	107249
C_{15}	Rules and regulations for master of science	19583	C_{25}	A translated novel	249782
C_{16}	Dictionary	23147	C_{26}	A book	355475
C_{17}	A paper on simulating software quality	44731	C_{27}	A Novel	403029
C_{18}	A report on e-government	124936	C_{28}	A Novel	561651
C_{19}	Higher education law	193943	C_{29}	A Novel	696351
C_{110}	Report on activities for electronic transformation	325267	C_{210}	A translated novel	726431

Table 4. Compression percentage and computation times (file size is sorted in ascending order).

File #	**Comp. Percentage**			**Comp. Time (msc.)**			**File #**	**Comp. Percentage**			**Comp. Time (msc.)**		
	Algorithm			**Algorithm**				**Algorithm**			**Algorithm**		
	SA	**Huff.**	**LZW**	**SA**	**Houf.**	**LZW**		**SA**	**Houf.**	**LZW**	**SA**	**Houf.**	**LZW**
C_{11}	13.00	33.80	35.15	744	24	13	C_{24}	36.62	35.87	48.54	2958	329	358
C_{12}	12.87	33.52	39.39	777	64	28	C_{18}	36.05	33.41	48.10	3618	379	230
C_{13}	17.97	32.71	41.33	410	63	32	C_{19}	43.22	35.94	51.39	846	622	348
C_{14}	23.41	31.46	42.94	1093	55	69	C_{25}	34.95	34.28	45.38	7115	812	486
C_{15}	30.62	33.12	46.34	4634	87	35	C_{110}	39.85	32.44	52.62	1647	54	515
C_{16}	15.86	35.50	42.13	806	93	94	C_{26}	30.05	33.51	39.85	12629	1204	768
C_{21}	24.06	34.11	43.52	1319	172	147	C_{27}	36.33	35.54	47.02	10963	1456	749
C_{17}	24.04	31.09	44.01	223	121	83	C_{28}	35.55	33.80	46.07	15705	1512	1041
C_{22}	30.09	31.70	54.65	1373	252	88	C_{29}	32.29	35.92	42.42	4634	1677	1291
C_{23}	31.30	33.00	46.47	2236	229	278	C_{210}	37.27	35.01	45.03	1373	2852	1363

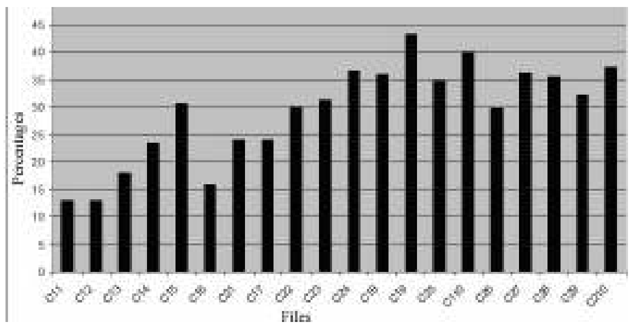

Figure 1. Compression percentage for SA algorithm for different files.

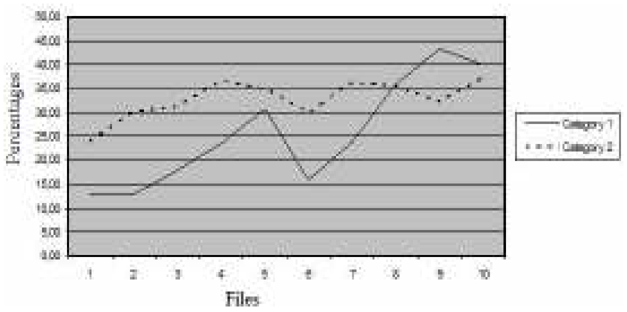

Figure 2. Compression percentage of two categories.

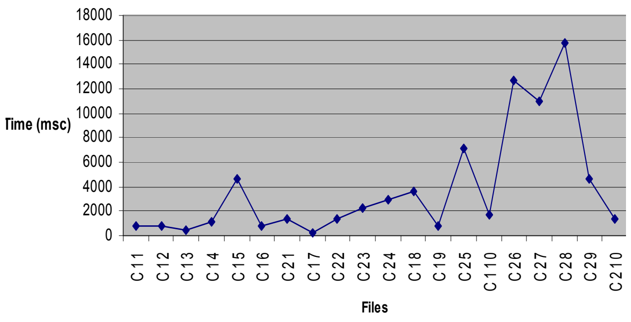

Figure 3. Compression time (file size is sorted in ascending order).

7. Discussion

Present study provides several notable contributions. First, our algorithm is robust against corruption in compressed files. This means, if any corruption occurs when compressing a syllable, it results in a typo only for that syllable as a result of decompression. The rest of the text stays uncorrupted. This property is superior to many other popular algorithms because bit errors are catastrophic for most of the other algorithms and result in an unreadable file after the point of corruption. Second, the proposed algorithm, contrary to many other algorithms [15], is modular and each module is clearly defined. Therefore, the entire system is simple, easy to understand, implement and make modifications. Third, SA needs less memory space due to using volatile dictionary. Finally, it extends theoretical insights from other algorithms reported in the literature and our results may provide a basis for discussions and extensions regarding the use of languages' syllabic characteristics in text compression.

Currently, two strings with different capitalizations are treated as two different words, but they could be combined into a single entry and end-of sentence test could predict which form to use. Using one or more non-volatile dictionaries for syllables and/or considering statistical/arithmetic coding of syllables may improve the performance of the syllabic algorithm in terms of both compression percentage and compression time. Additionally, a study on performance comparison with other multi-syllabic languages (or mixed languages) will shed light on the reaction of the proposed algorithm against different language structures.

The proposed algorithm has also some advantages compared to the method reported by Lansky and Zemlicka [17, 18, 19]. In their study, Lansky and Zemlicka [17] focused on the specification of syllables. They created two syllable-based compression algorithms and, as they noted, their algorithm has problems in decomposing the words into syllables (page 39). To improve the compression, they created a database of frequent words, which of course makes the procedure more complicated. Additionally, the content of database may change from one language to another and, more importantly, is likely to be subjective even in the same language. As they also expected, the databases can improve compression ratio for smaller documents. The experimental results of their algorithms are ambiguous for Czech. Our algorithm, as noted before, uses a volatile dictionary only and this idea applies to all syllabic languages. This dictionary is created automatically and progressively for each text when the compression starts. On the contrary of Lansky and Zemlicka [17], the present approach performs better for larger files.

Ucoluk *et al.* [25] proposed a genetic algorithm approach for syllable based text compression. Their approach is based on Huffman encoding upon mixed alphabet of characters and syllables. Although, ideally, this approach requires extensive Huffman tree constructions, the authors used a theoretical approximation for estimating the compressed length using mathematical operations. In addition, rare syllables are dissolved into characters every time in Ucoluk's [25] approach, in which case the problem is that which syllables should be included to ensure the optimal length of the compressed text [21]. Furthermore, the proposed approach generally produced better compression percentages (30-43%) than the genetic algorithm of Ucoluk *et al.* [25].

Another genetic algorithm for syllable based text compression has been proposed by Lansky and Khutan [21]. They obtained dictionaries using genetic algorithm and applied on texts with different languages such as Czech and English. Compared to Lansky and Kuthan [21], our approach performs better for especially medium and large files in terms of compression percentages.

Finally, some authors also worked on compression techniques for Turkish documents. For example, Celikel *et al.* [7] proposed a secure compression (SeCom) algorithm. They stressed for the need of security during compression. To this end, they applied multiple *encoding* to strengthen the security of the *SeCom* scheme. In another study, Celikel *et al.* [6] proposed Word-Based Fixed and Flexible List Compression technique. Diri [10] proposed a method for lossless compression for monograms, diagrams, trigrams, root grams and suffixes individually using a statistical approach. Their experiments showed that compression ratio is changing from 39% to 59%. Actually, the compression technique for Turkish language is firstly proposed by one of the authors of the present paper [1]. This technique was based on partitioning the word into its root and suffixes by using dictionaries. The reported compression goes up to 47% with this approach. The reader should note here that all the works discussed in this paragraph belong to word based compression and do not use syllables.

8. Conclusions

The proposed lossless text compression algorithm takes syllabic characteristic of multi-syllabic languages into consideration for compressing a given text. The components of the algorithm are: source file, filtering unit, syllables unit, compression unit, dictionary file and target file. The method uses variable bit length representation depending on the number of different syllables in the dictionary and performs compression in three steps. The first step is filtering to find non-alphabetic characteristics. The second step uses an automaton to partition the words into their syllables. Finally, the compression unit first creates a volatile dictionary to identify bit representations for different syllables and then uses this dictionary to develop the target file which contains compressed text. With these features the proposed technique can be a valuable contribution in the field of text compression.

The proposed approach was implemented in Turkish language and experiments were conducted using 20 different and reasonably selected texts whose sizes vary between 4.6 and 725 Kbytes. The compression rates were observed to change from 13.0% to 43.2%. Experiences indicated that higher compression rates were achieved with increasing text sizes.

Acknowledgement

We used a modified version of the finite automaton given by Asliyan *et al.* [2]. The modifications were done by Dr. Hurevren Kilic of Atilim University. Authors are thankful to them.

References

[1] Akman I., "A New Text Compression Technique Based on Language Structure," *Journal of Information Science*, vol. 21, no. 2, pp. 87-94, 1995.

[2] Asliyan R., Günel K., and Filiz A., "Otomatic Syllable System for Turkish and Syllable Statistics," *in Proceeding of Academic Informatics, Information Technologies Conference IV*, Pamukkale University, 2006.

[3] Buyukkuscu I. and Adali E., "Developing Roots Using Syllables," *Computer Sciences and Engineering Journal*, vol. 2, pp. 25-29, 2006.

[4] Blandon J., Adjouadi M., and Emami S., "A Synergistic Text Compression Method STCM," *in Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing*, vol. 3, pp. 2773-2776, Florida, 2002.

[5] Cahill L., "Syllable-Based Morphology," *in Proceedings of Computational linguistics*, vol. 3, pp. 48-53, United Kingdom, 1990.

[6] Celikel E., Dalkilic M., and Dalkilic G., "Word-Based Fixed and Flexible List Compression," *Lecture Notes in Computer Science*, vol. 3733, pp. 780-790, 2005.

[7] Celikel E. and Dalkilic M., "Experiments in a Secure Compression Algorithm, Word-Based 790, Fixed and Flexible List Compression," *Lecture Notes in Computer Science*, vol. 3733, pp. 780-2005.

[8] Cormack G. and Horspool R., "Data Compression Using Dynamic Markov Modelling," *The Computer Journal*, vol. 30, no. 6, pp. 541-550, 1987.

[9] Comrie B., *The World's Major Languages*, Croom Helm, London, 1987.

[10] Diri B., "Content Based Compression of Turkish Documents," *Pakistan Journal of Applied Science*, vol. 1, pp. 446-451, 2001.

[11] Dvorsky J., Pokorny J., and Snasel V., "Word-Based Compression Methods for Large Text Documents," *in Proceeding of IEEE Data Compression Conference*, pp. 523, USA, 1999.

[12] http://www.turkishembassy.si/Turkish_Language .htm, last visited February, 2009.

[13] Horspool R., "Improving LZW," *in Proceeding of Data Compression Conference*, pp. 332-341, Utah, 1991.

[14] Horspool R. and Cormack G., "Constructing Word-Based Text Compression Algorithms," *in Proceedings of IEEE Second Data Compression Conference*, pp. 62-81, 1992.

[15] Kristensen M., "GZip vs. Deflate: Compression and Performance," Google Analytics, available at: http://www. webpronews. com/ expertarticles /2006/12/08/gzip -vs -deflate- compression-and performance, 2006.

[16] Katsiaryna C., Lansky J., and Galambos, L., "Syllable-Based Compression for XML Documents," *in Proceedings of Czech Republic*, pp. 21-31, Desna, 2006.

[17] Lansky J. and Zemlicka M., "Text Compression: Syllables," *in Proceedings of DATESO' 2005*, pp. 32-45, Prague, 2005.

[18] Lansky J. and Zemlicka M., "Compression of Small Text Files Using Syllables," *Technical Report*, Department of Software Engineering, Faculty of Mathematics and Physics, Prague, 2006.

[19] Lansky J. and Zemlicka M., "Compression of Small Text Files Using Syllables," *in Proceedings of Data Compression Conference* p. 458, Utah, 2006.

[20] Lansky J., Chernik K., and Vlckova Z., "Syllable-Based-Burrows-Wheeler Transform," *in Proceedings of the Dateso, Annual International Workshop on Databases, Texts, Specifications and Objects*, vol. 235, pp. 1-10, Desna, 2007.

[21] Lansky J. and Kuthan T., "Genetic Algorithms in Syllable Based Text Compression," *in Proceedings of DATESO*' 2007, pp. 21-34, 2007.

[22] Moffat A., "Word Based Text Compression," *Software: Practice and Experience*, vol. 19, no. 2, pp. 185-198, 1989.

[23] Sestak R. and Lansky J., "Compression of Concatenated Web Pages Using XBW," *in Proceedings of SOFSEM' 2008,* vol. 4910, pp. 743-754, *LNCS,* 2008.

[24] Skibinski P., "Two-Level Directory Based Compression," *in Proceedings of IEEE Data Compression Conference*, pp. 481, 2005.

[25] Ucoluk G. and Toroslu H., "Genetic Algorithm Approach for Verification of the Syllable Based Text Compression Technique," Computer *Journal of Information Science*, vol. 23, no. 5, pp. 365-372, 1997.

[26] Ziv J. and Lempel A., "A Universal Algorithm for Sequential Data Compression," *IEEE Transactions on Information Theory*, vol. 23, IT-24, pp. 337-343, 1997.

Ibrahim Akman received his PhD in operations research from Lancaster University, UK in 1984. He has served on the editorial boards of Electronic Journal of e-Government and International Journal of Information Technology and Management (IJITM). His research interests include software engineering, simulation, software piracy, e-government, human resource management, and data compression.

Hakan Bayindir received his Bachelor degree in computer engineering from Atilim University, Turkey in 2007. Currently, he is continuing his Master degree from Atilim University and working at High Performance and Grid Computing Center which is a part of Turkish Academic Network Information Center (TUBITAK-ULAKBIM) as a Linux system administrator and developer. His research interests include multi-agent systems, parallel programming, and AI.

Serkan Ozleme received his Bachelor degree in 2007 from Computer Engineering Department of Atilim University, Turkey. He has received several certificates about programming using Microsoft technologies and relevant tools. Currently, he is working as software engineer in system integration business as an application developer which integrates to various web services and applications.

Zehra Akin received the BS degree in computer engineering from Atilim University in 2007, Turkey. She is working as a specialist software developer at the Radio and Television Supreme Council (RTÜK) ERP project (METEKSAN System).

Sanjay Misra is an assistant professor in Department of Computer Engineering, Atilim University, Ankara, Turkey. Presently, he is working in the area of software engineering, especially on software quality estimation through software metrics. His area of interests are software measurement, verification and validation techniques, object oriented technologies, data compression, XML, Web Services, and cognitive informatics.