http://ansinet.com/itj

ITJ

ISSN 1812-5638

INFORMATION TECHNOLOGY JOURNAL

ANSInet
Asian Network for Scientific Information
308 Lasani Town, Sargodha Road, Faisalabad - Pakistan

Information Technology Journal 9 (5): 1013-1021, 2010
ISSN 1812-5638
© 2010 Asian Network for Scientific Information

Fast Text Compression Using Multiple Static Dictionaries

A. Carus and A. Mesut
Department of Computer Engineering, Trakya University, Edirne, Turkey

Abstract: We developed a fast text compression method based on multiple static dictionaries and named this algorithm as STECA (Static Text Compression Algorithm). This algorithm is language dependent because of its static structure; however, it owes its speed to that structure. To evaluate encoding and decoding performance of STECA with different languages, we select English and Turkish that have different grammatical structures. Compression and decompression times and compression ratio results are compared with the results of LZW, LZRW1, LZP1, LZOP, WRT, DEFLATE (Gzip), BWCA (Bzip2) and PPMd algorithms. Our evaluation experiments show that: If speed is the primary consideration, STECA is an efficient algorithm for compressing natural language texts.

Key words: Text compression, dictionary based compression, multiple dictionary, static dictionary, diagram coding

INTRODUCTION

Lossless compression coding techniques are generally classified into two groups: statistical-based coding and dictionary-based coding. In statistical-based coding, compression takes place based on using shorter codewords for symbols that occur more frequently. Because all the symbols are not coded with same bit-length, this method is also known as variable length coding. The two most well known statistical-based coding techniques are; Huffman Coding and Arithmetic Coding (Moffat *et al*., 1995). PPM (Prediction by Partial Matching) is another form of statistical coding, which predicts the next symbol based on a sequence of previous symbols (Cleary and Witten, 1984). The variants of this technique such as PPMd (Shkarin, 2002) achieve the best compression ratios for nearly all type of data. Dictionary-based coding techniques replace input strings with a code to an entry in a dictionary. The most well known dictionary-based techniques are Lempel-Ziv Algorithms (Ziv and Lempel, 1977, 1978) and their variants (Storer and Szymanski, 1982; Welch, 1984). Dictionary-based algorithms are generally combined with a variable length coding algorithm to improve compression ratio. For example DEFLATE algorithm (Deutsch, 1996) is a combination of the LZ77 Algorithm and Huffman Coding. It is also possible to increase compression ratio by using a transformation method before compression (Fauzia and Mukherjee, 2001; Burrows and Wheeler, 1994; Moffat and Isal, 2005; Robert and Nadarajan, 2009).

Dictionary-based techniques can be divided into three categories. In a static dictionary scheme, a suitable dictionary is used for all compression tasks. In semi-static dictionary scheme, an optimal dictionary is prepared in the first-pass using the character distribution in the source and compression of the source data is performed in the second-pass by using this dictionary. In an adaptive dictionary scheme, the dictionary is generated and continuously updated during the compression process. The static dictionary scheme is only appropriate when considerable prior knowledge about the source is available. If there is not sufficient prior knowledge about the source, using adaptive or semi-static schemes is more effective.

Some adaptive dictionary coding methods (Hoang *et al*., 1999; Nakano *et al*., 1994), which use multiple dictionaries, have attempted to improve the compression of dictionary-based methods without slowing them too much. The idea is to use different dictionaries according to the last coded character(s). The decoder can easily decide which dictionary to use, by looking at the last coded character(s). Using multiple dictionaries can increase the compression ratios of LZ-family algorithms up to 15% (Hoang *et al*., 1999).

Although, static dictionary methods are language dependent when they are used with texts written in natural languages, they have speed and simplicity advantage (Bell *et al*., 1989). We focused on how the compression ratio and compression/decompression times are affected when multiple static dictionaries are used in text compression. The aim of our STECA algorithm (Static Text Compression Algorithm) is to reach acceptable compression ratio in a very low compression time. We were also trying to explore to see whether the

Corresponding Author: Aydin Carus, Trakya Universitesi, Prof. Dr. Ahmet Karadeniz Yerleskesi, Muhendislik-Mimarlik Fak., Bilgisayar Muhendisligi Bol., Edirne, Turkiye Tel: +90 284 226 12 17 Fax: +90 284 226 12 25

compression ratio and time results of two different languages such as English and Turkish are similar to each other or not. The language dependency of STECA seems to be a disadvantage, but in order to improve compression time or compression ratio it can sometimes be necessary to use a language dependent approach. Many language dependent transformation and compression methods are developed in recent years (Nakamura *et al*., 2009; Skibiński *et al*., 2005; Skibiński and Swacha, 2009; Teahan and Cleary, 1997).

THE MULTIPLE STATIC DICTIONARIES APPROACH

STECA algorithm performs compression with using most frequently used digrams and trigrams instead of ASCII codes that are not used or infrequently used in text files. In diagram coding, at each coding step the next two characters are inspected to see if they correspond to a diagram in the dictionary. If so, the corresponding index in the dictionary is encoded; otherwise only the first character is encoded. The coding position is then shifted by one or two characters as appropriate. If three characters are used instead of two, the coding algorithm is named as trigram coding.

Dictionary generation and hash tables: In order to generate a static dictionary for a language, frequently used characters of that language must be known. An alphabet Σ is generated with n frequently used characters and a new code table is prepared for Σ. Besides these n characters, an escape character (which is used when the current encoded character is not in the code table) is also defined. We select 0 to define escape character for the code table. Thus, after separation of this character, m = 256-n-1 codes left to represent digrams and trigrams. Over 150 MB of novels, daily newspaper articles, social and scientific articles are collected for both Turkish and English; and these collections are used to generate static dictionaries of these languages. The results of our studies with Turkish and English collections show that: the compression ratio is best when n = 95 and m = 160.

The main idea in STECA is to select a particular dictionary by looking the last encoded/decoded character. Using multiple dictionaries instead of a single dictionary provides better compression ratio, because this approach increases the finding probability of digrams and trigrams in the dictionary. The most frequently used 160 digrams/trigrams and total numbers/percentages of their occurrence in our English text collection are given in Table 1 and 2. The same values are also calculated for digrams/trigrams which are followed the letters: a, b and c, these values are shown in Table 3 and 4. The percentages of the first 160 digrams/trigrams in Table 3 and Table 4 are higher than the ones in Table 1 and 2 and

Table 1: The most frequently used digrams in English collection

Index	Diagram	Repeat No.	Sum	%
1	e	4,646,627	123,754,525	78.68%
2	t	3,677,964		
…	… …	…		
160	d o	229,179		
…	… …	…	33,531,874	21.32%
6.429	t h	1		

Table 2: The most frequently used trigrams in English collection

Index	Trigram	Repeat No.	Sum	%
1	t h	2,425,398		
2	t h e	1,996,736		
…	… … …	…	55,365,159	35.20%
160	n o t	161,253		
…	… … …	…	101,921,239	64.80%
72,089	d c i	1		

Table 3: The most frequently used digrams after a, b and c in English collection

	Index	Diagram	Repeat No.	Sum	%
	1	n d	784,379		
	2	t	535,767	9,045,391	92.4%
a	…	… …	…		
	160	d l	8,354		
	…	… …	…		
	1,684	n s	1	744,671	7.6%
	1	e	135,221		
	2	y	109,891		
b	…	… …	…	1,636,205	98.8%
	160	n	286		
	…	… …	…		
	770	m g	1	20,167	1.2%
	1	e	218,500		
	2	h	171,495		
c	…	… …	…	3,104,181	97.7%
	160	h l	1,135		
	…	… …	…		
	1,041	t g	1	72,917	2.3%

Table 4: The most frequently used trigrams after a, b and c in English collection

	Index	Trigram	Repeat No.	Sum	%
	1	n d	678,179		
	2	t i o	157,118		
a	…	… … …	…	5,291,983	54.05%
	160	r k	12,150		
	…	… … …	…	4,498,079	45.95%
	15,039	b f a	1		
	1	u t	88,221		
	2	e e n	68,502		
b	…	… … …	…	1,274,913	76.97%
	160	a t	1,931		
	…	… … …	…	381,459	23.03%
	5,039	e e j	1		
	1	o m p	59,754		
	2	a n	52,998		
c	…	… … …	…	2,186,654	68.83%
	160	l e s	4,637		
	…	… … …	…	990,444	31.17%
	6,607	e v s	1		

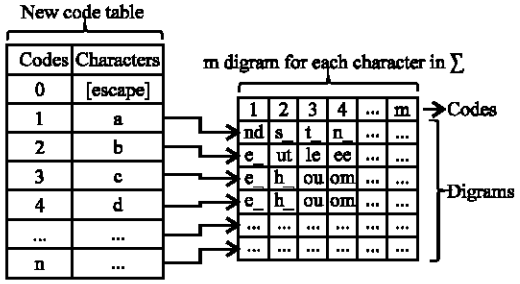

Fig. 1: The dictionary in Mode D for english

this result shows that: the compression ratio can be improved if compression is made with using digrams/trigrams which are grouped according to the characters before them when m = 160.

STECA has three different modes. Mode D uses only digrams, Mode T uses only trigrams and Mode DT uses both digrams and trigrams in the dictionary. The structure of the generated dictionary in Mode D for English language is shown in Fig. 1. The dictionary structures of the other modes are similar to Mode D.

As seen from Fig. 1, the total dictionary size is equal to the size of white blocks, which is equal to n + 2(n×m) bytes (In Mode T it is equal to n+3 (n×m) bytes). STECA uses hash tables for encoding the indexes of digrams or trigrams in the dictionary. If the diagram or trigram being encoded is not in the hash table, 0 will be encoded. The values that are not 0 in these hash tables are stored in the code of compression program to represent the dictionary. The hash functions that are used to access to these hash tables are given below:

Hash function for Mode D is;

$$\text{HashD}(l, x, y) = \begin{cases} \text{dictionary index of the digram} & \text{if digram} \in \text{dictionary} \\ 0 & \text{if digram} \notin \text{dictionary} \end{cases}$$

Hash function for Mode T is;

$$\text{HashT}(l, x, y, z) = \begin{cases} \text{dictionary index of the trigram} & \text{if trigram} \in \text{dictionary} \\ 0 & \text{if digram} \notin \text{dictionary} \end{cases}$$

Where:

l = A code from new code table for last encoded character
x = A code from new code table for first character
y = A code from new code table for second character
z = A code from new code table for third character

Mode DT uses one or two HashT functions. If the trigram is an element of the dictionary, the index of the trigram in the dictionary is returned from first HashT. If the trigram is not an element of the dictionary, 0 is returned and a second HashT function is used with z = n+1 to search for diagram.

Using hash tables is the fastest solution of finding 'xy diagram or xyz trigram is an element of the dictionary or not' problem, but they are expensive in terms of memory usage. The required memory for hash tables are n^3 (837.3 kb when n = 95) for Mode D and n^4 (77.7 Mb when n = 95) for both Mode T and Mode DT. As mentioned earlier, frequently used characters of a language must be known in order to generate a static dictionary for that language. Individual alphabets that have n elements are prepared for English and Turkish with using these frequently used characters. If the n value is decreased, the required memory for hash table will also be decreased. However, if the n value is increased, the escape character will be used less frequently and the compression ratio will be better.

The 95 (n) individual static dictionaries are generated for each character in Σ and each of them contains 160 (m) digrams in Mode D or 160 trigrams in Mode T. In Mode DT, the m value must be divided into two pieces for digrams and trigrams. After performing many compression tests it was seen that: the compression ratio was not changed too much when the numbers of digrams and trigrams in m were selected differently. For this reason, we decided to give the same size (80) for both of them.

Compression algorithm: The encoding algorithm of Mode T is given in Fig. 2 and it is explained in this section with an example. Mode D is very similar to Mode T and Mode DT is a combination of these two modes.

Suppose that we want to compress john@xmail.com mail address with Mode T. Before compression process takes place, the generated dictionary is loaded into memory and NULL value is assigned into LEC (last encoded character). The first character of the source, which is j, is searched in Σ by using code table. It is found in Σ and the index (10) is encoded. A dictionary of the letter j is chosen for coding. The second character of the source, which is o, is searched in Σ. It is found and assigned as the first character of trigram. Similarly, the third and the fourth characters of the source, which are h and n, are found in Σ and they are assigned as the second and third characters of the trigram. The trigram ohn is then searched in the dictionary of the letter j. suppose that the trigram is found in 5th index of this dictionary. This index (5) is encoded and the third character of the trigram (n) is assigned into LEC. To form the new trigram, another character is taken from the source, which is @ and searched in Σ. Suppose that it is not found in Σ. To encode @, first the escape character (0) is encoded and later ASCII code of this character (64) is encoded. The index in the code table of the next character in the source, which is x, is encoded (24) and it is assigned into LEC.

```
Open source and destination files
LEC = Read a character from source file
codetable index of LEC is encoded
trigram count = 0
While not end of source file {
    chr = Read a character from source file
    if chr exists in S {
        trigram[trigram count] = chr
          trigram count = trigram count + 1
          if trigram count = 3 {
            if the trigram exists in the dictionary of LEC {
                    The corresponding dict. index is encoded
                    LEC = trigram[2]
            }
            else {
                    trigram[0] is encoded
                    LEC = trigram[0]
                characters of the trigram are shifted from
                right to left
                trigram count = trigram count - 1
            }
        }
    }
     else {
        if trigram_count > 0 {
               LEC = trigram[trigram count-1]
               characters of trigram are encoded
               trigram count = 0
               trigram = NULL
        }
          0 (codetable index of the escape character) is
encoded
          ASCII code of chr is encoded
    }
}
Close source and destination files
```

Fig. 2: STECA/mode-T encoding algorithm

The next three characters (mai), which are found in Σ, are assigned into trigram. Suppose that this trigram is not found in the dictionary of the letter x. The index of m in the code table, which is 13, is encoded and it is assigned into LEC. After the shifting process the new trigram becomes as ail. Suppose that this trigram is found in the dictionary of the letter m. The index of the dictionary is encoded to perform the compression. The third character of the trigram (l) is assigned into LEC and the next 3 characters in the source (.co) are assigned to the trigram. Suppose that this trigram is not found in the dictionary of l. The index of '.' in the code table, which is 27, is encoded and it is assigned into LEC. After the shifting process the new trigram becomes as com. Suppose that this trigram is found in the dictionary of the letter '.' and the index of the dictionary is encoded.

In Mode DT, firstly, the trigram is searched in trigram part of the dictionary relating to LEC. If it is found, it is encoded as explained above. If it is not found, the first and the second characters are combined and generated diagram is searched in the diagram part of the dictionary relating to LEC. If it is found, a similar encoding process is performed. But this time, the middle character of the trigram is assigned into LEC and the third character of the trigram becomes the first character of the new trigram.

EVALUATION METHODOLOGY

Other algorithms used for comparison: STECA algorithm was compared with the following dictionary-based and statistical-based algorithms: LZW, LZRW1, LZP1, LZOP, WRT, DEFLATE (Gzip), BWCA (Bzip2) and PPMd. Brief descriptions of these algorithms are given in this section.

LZW: Abraham Lempel and Jacob Ziv created a compression scheme in 1978 (Ziv and Lempel, 1978), which works by entering phrases into a dictionary and then, when a reoccurrence of that particular phrase is found, outputting the dictionary index instead of the phrase. Several algorithms are based on this principle, which are known as LZ78 family, differing mainly in the manner in which they manage the dictionary. The most well known modification of LZ78 Algorithm is Terry Welch's LZW Algorithm (Welch, 1984). The LZW

compression algorithm is more commonly used to compress binary data, such as bitmaps. UNIX compress and the GIF and TIFF image compression formats are both based on LZW. The C code of LZW that was used in our comparison was developed by Mark Nelson (Nelson and Gailly, 1995).

LZRW1: This compression algorithm was created by Williams (1991). It uses the single pass literal/copy mechanism of LZ77. Different from LZ77, it uses a simple hash table mechanism that causes fast compression and decompression with a cost of some compression ratio. LZRW2 and LZRW3 algorithms are similar to the LZRW1 algorithm. They are slower than LZRW1 but they can compress better.

LZP1: The LZP algorithm is a technique that combines PPM-style context modeling and LZ77-style string matching. This fast and efficient algorithm was developed by Bloom (1996). The LZP finds the most recent occurrence of the current context and compares the string following that context to the current input. The length of match between the two strings is found and then coded. If the length of match is zero, a literal (single non-matched byte) is written using another method (such as lower-order LZP, or PPM). Bloom presented four different implementations of LZP (1 to 4-fastest to slowest) in his paper. We used LZP1 in our comparison.

LZOP: LZOP was developed by Oberhumer (1997) and after that year many revisions have been made and new versions were released by him. LZOP uses the LZO data compression library, which is also developed by Oberhumer. LZO is a portable lossless data compression library written in ANSI C and offers pretty fast compression and extremely fast decompression. LZOP is very similar to Gzip and its main advantages over Gzip are much higher compression and decompression speed (at the cost of some compression ratio). It also includes slower compression levels achieving a quite competitive compression ratio while still decompressing at this very high speed. LZOP Version 1.02 was used in our comparison.

WRT: WRT (Word Replacing Transformation) is an English text preprocessor that replaces words in input text file with shorter codewords and uses several other techniques to improve performance of latter compression. It was developed by Skibiński *et al*. (2005). In terms of compression, it brings an improvement on the order of 81-2% in comparison with ordinary PPM or BWCA compressors and even about 21-27% with LZ77 (Lempel-Ziv) compressors. We used WRT Mode 0, which is LZ77 based, in our comparison.

Bzip2: Bzip2 compresses files using the Burrows-Wheeler block-sorting text compression algorithm and Huffman coding. The old version of Bzip2 (Bzip) used arithmetic coding after the block sort. Using arithmetic coding was discontinued because of the patent restrictions. Although this algorithm is extremely slow at compressing, compression ratio is generally better than that achieved by more conventional LZ77/LZ78-based compressors and close to the performance of the PPM family of statistical compressors. Bzip2 version 1.0.4 was used in our comparison.

Gzip: Gzip uses DEFLATE algorithm which is a combination of the LZ77 algorithm and Huffman coding. It was originally defined by Phil Katz for version 2 of his PKZIP archiving tool and was later specified in RFC 1951 (Deutsch, 1996). DEFLATE is widely thought to be free of any subsisting patents and at a time before the patent on LZW (which is used in the GIF file format) expired, this has led to its use in Gzip compressed files and PNG image files, in addition to the ZIP file format for which Katz originally designed it. Gzip version 1.2.4 was used in our comparison.

PPMd: PPM is an adaptive statistical data compression technique based on context modeling and prediction. The name stands for Prediction by Partial Matching. PPM models use a set of previous symbols in the uncompressed symbol stream to predict the next symbol in the stream. PPMd is an open-source data compression algorithm developed by Shkarin (2002). WinZip 10.0 and 11.0 uses Version I Revision 1 of the algorithm. We used a newer version, which is Version J Revision 1 (2006), in our comparison.

Test environment: The measurements of this work were calculated with a computer, which has Intel Core 2 Duo 1.83 MHz CPU, 2 GB main memory, 120 GB 7200 Rpm hard disk and Microsoft Windows XP Professional operating system. The C codes of LZW and LZRW algorithms that were used in this evaluation were compiled in Release mode of C Compiler of Microsoft Visual Studio. NET 2003 Edition.

Data sets: Two files were prepared for using in compression tests. The English text file, which is 16.453.299 bytes in size, is composed of 6 different text files. The size and descriptions of these files are given in Table 5, the files in this table were taken from Canterbury Corpus (Bell *et al*., 1989), Silesia Corpus (Deorowicz, 2003) and Project Gutenberg.

The Turkish text file, which is 15.828.594 bytes in size, is a simplified version of The Metu Corpus (Say *et al*.,

Table 5: The 6 different files in the English test file

File name	Size (bytes)	Description	Taken from
1musk12.txt	1.349.141	The Three Musketeers by Alexandre Dumas père	Gutenberg (Etext-No. 1257)
alice29.txt	152.089	Alice's Adventures in Wonderland by Lewis Carroll	Canterbury Corpus
anne11.txt	587.053	Anne of Green Gables by L. M. Montgomery	Gutenberg (Etext-No. 45)
asyoulik.txt	125.179	As You Like It by William Shakespeare	Canterbury Corpus
bible.txt	4.047.392	The King James version of the bible	Canterbury Corpus
dickens.txt	10.192.446	Collected works of Charles Dickens	Silesia Corpus

2002). Some characters and words that are belonging to XML format were eliminated to obtain only the simple text.

RESULTS

Compression ratio: The compression ratios of English and Turkish language text files are given in Fig. 3 from best to worst. The compression ratios that are given in bits/character (bpc) were calculated as:

$$\text{Compression ratio} = \frac{\text{Compressed file size}}{\text{File size}} \times 8 \qquad (1)$$

We can see in Fig. 3 that PPMd is the best algorithm in compression ratio, Bzip takes the second and Gzip takes the third place. The worst algorithm is LZRW1. The compression ratios of all modes of the STECA algorithm are worse than 4 algorithms (PPMd, Bzip, Gzip and WRT) and better than the other 4 algorithms (LZW, LZRW1, LZOP and LZP1). The best mode of STECA in compression ratio is Mode DT. In this mode STECA can compress English and Turkish test files into more than half of their original size.

When we look at differences between compression ratios of English and Turkish text files, we can see that in average English is nearly 4.5% more compressible than Turkish.

Compression and decompression speed: To evaluate compression and decompression speed of all modes of the STECA algorithm and the other compression algorithms, both test files were compressed and decompressed 6 times with each algorithm and the average values were taken. Obtained average compression speed values are given in Fig. 4 and average decompression speed values are given in Fig. 5 as megabits/second in decreasing order.

It can be seen from Fig. 4 that STECA-D is the fastest compression algorithm. The compression speed of STECA-D is 433 Mbps in English and 390 Mbps in Turkish. The compression speed of STECA-T and the compression speed of LZOP are nearly equal to each other. The former is better in English and the latter is better in Turkish. STECA-DT takes the fourth place in our compression tests. Bzip is the slowest compressor for both English and Turkish.

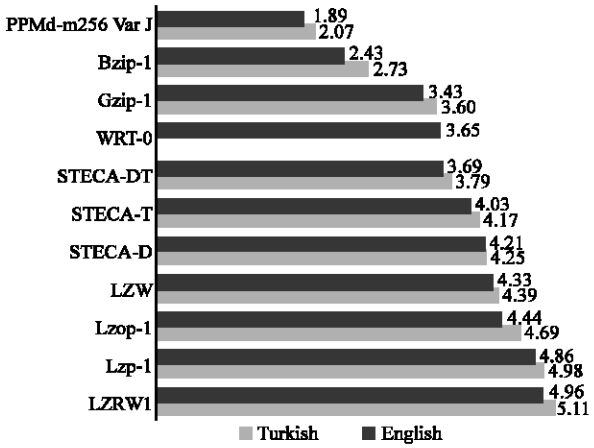

Fig. 3: Compression ratios of the algorithms (bpc)

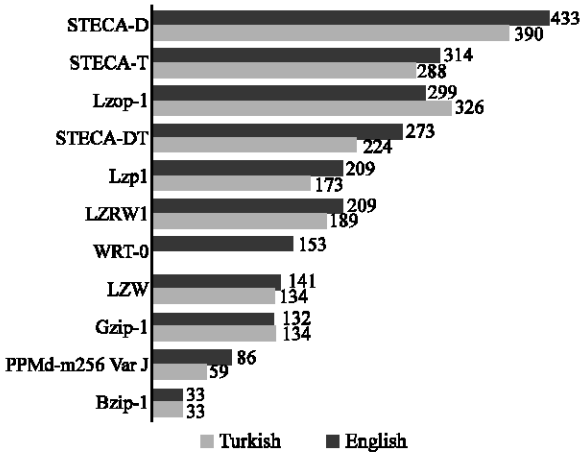

Fig. 4: Compression speeds of the algorithms (Mbps)

The decompression results, which are given in Fig. 5, are a little bit different from compression results. LZOP achieves the best result while Mode D, Mode T and Mode DT of STECA algorithm are in the second, third and fourth places respectively. The decompression speeds of LZW, LZOP, Gzip, Bzip and STECA algorithms are faster than their compression speeds, while the decompression speeds of LZRW, WRT, LZP and PPMd are nearly equal or slower than their compression speeds. PPMd is the slowest decompressor for both English and Turkish.

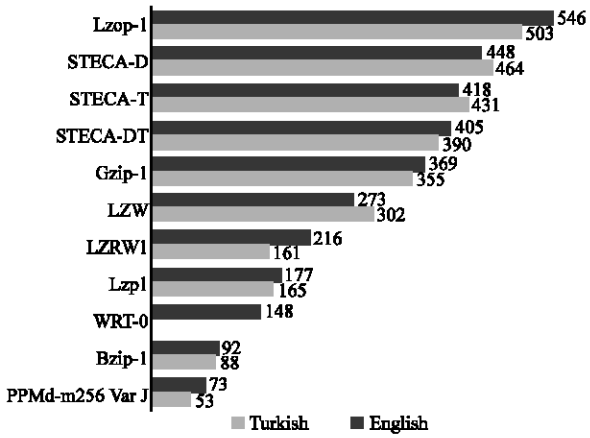

Fig. 5: Decompression speeds of the algorithms (Mbps)

It can be seen from Fig. 3-5 that the sort order of the STECA Modes are the same for both compression and decompression and this sort order is the reverse of the one in the compression ratio. Generally this situation is the same for all compression algorithms: the fastest mode of a compression algorithm is the worst mode in compression ratio. We always used the fastest modes of the other algorithms, because in this comparison we were focused on the speed, not the ratio.

Evaluation of the results for network transmission: When a compression algorithm is used to speed up a network transmission, compression ratio, compression speed and decompression speed of this algorithm are all important parameters. Compression and decompression speeds are more important in fast networks, while compression ratio is more important in slow networks. For example; an IP packet, which is 1000 bytes in size, can be sent to the receiver in 0.08 msec over a 100 Mbps network. If this packet can be compressed to half of its size, it can be sent in 0.04 msec after compression. If the compression and decompression times are smaller than 0.04 seconds in total, making compression for this transmission is suitable. If the bandwidth is 10 Mbps, the same packet can be sent to the receiver in 0.8 msec when it is not compressed and 0.4 msec when it is compressed. Thus, when the bandwidth is 10Mbps, the compression and decompression times which are less than 0.4 msec in total can be acceptable. We can say that if an algorithm is slow in compression and decompression, it is suitable only in slow networks.

The maximum suitable bandwidth for a particular compression algorithm can be calculated by using compression ratio, compression speed and decompression speed values. It is obvious that the transmission time (t) is:

$$t = \text{Compression time} + \text{Send time} + \text{Decompression time}$$

$$t = \frac{\text{packet size}}{\text{com. speed}} + \frac{\text{compressed packet size}}{\text{network speed}} + \frac{\text{packet size}}{\text{decomp. speed}} \quad (2)$$

This t value must be smaller than the send time of the uncompressed IP packet:

$$\frac{\text{packet size}}{\text{comp. speed}} + \frac{\text{comp. packet size}}{\text{network speed}} + \frac{\text{packet size}}{\text{decomp. speed}} < \frac{\text{packet size}}{\text{network speed}}$$

$$\frac{\text{packet size}}{\text{comp. speed}} + \frac{\text{packet size}}{\text{decomp. speed}} < \frac{\text{packet size} - \text{comp. packet size}}{\text{network speed}}$$

$$\text{Network speed} < \frac{\text{packet size} - \text{comp. packet size}}{\left(\frac{\text{packet size}}{\text{comp. speed}} + \frac{\text{packet size}}{\text{decomp. speed}}\right)}$$

$$\text{Network speed} < \frac{1 - \frac{\text{comp. packet size}}{\text{packet size}}}{\frac{1}{\text{comp. speed}} + \frac{1}{\text{decomp. speed}}} \quad (3)$$

From Eq. 1, the compressed packet size can be calculated as:

$$\text{Compressed packet size} = \frac{\text{Compression ratio}}{8} \times \text{Packet size} \quad (4)$$

By using Eq. 4, 3 will become:

$$\text{Network speed} < \frac{1 - \frac{\text{Compression ratio}}{8}}{\frac{1}{\text{Comp. speed}} + \frac{1}{\text{Decomp. speed}}} \quad (5)$$

Maximum network speeds that prove Eq. 5 are calculated and shown in Fig. 6. As seen from Fig. 6, STECA-D is the most suitable algorithm for fast network transmissions. STECA-T, STECA-DT and LZOP algorithms are also convenient for 54 Mbps wireless communication. However, if compression and decompression speeds that are related to speeds of compressor and decompressor computers/devices are changed, the results in Fig. 6 are also changed. The results in Fig. 4-6 are dependent on the test environment. Therefore, if slower computers/devices are used for compression and decompression, the effect of these algorithms to network transmission will be decreased.

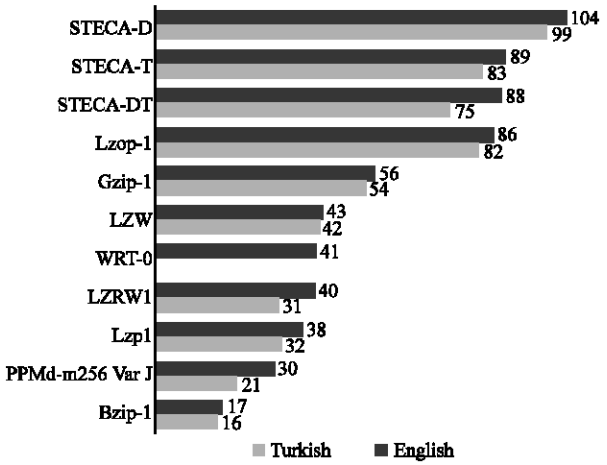

Fig. 6: Maximum suitable bandwidths of the algorithms (Mbps)

CONCLUSION

STECA algorithm is a good alternative to other compression methods especially for digital libraries and online encyclopedias with its fast decoding feature. Mode D has advantages of better compression and decompression speeds and lower memory usage and it can be able to speed up 54Mbps wireless network communication when compressor and decompressor hardware are not slow. Especially in compression, Mode D is much faster than other two modes and the memory usage of this mode is nearly 1% of memory usage of other two modes. Mode T and Mode DT achieve better compression ratios than Mode D (Fig. 3). However, these modes need more memory and have low compression and decompression speeds (Fig. 4, 5).

The compression ratio of STECA can be improved by using two or more last encoded characters for selecting the appropriate dictionary instead of a single character. However, if hash tables are used to find the index of a diagram/trigram in the dictionary, the required memory for storing these hash tables becomes too large. If a search algorithm is used rather than direct access with hash tables, this time the compression speed is worse. We made this improvement with using binary search and saw that its compression speed and ratio values are nearly the same with values of Gzip. Since, Gzip is not language dependent, it can be said that: using STECA with binary search is not a convenient solution.

REFERENCES

Bell, T.C., I.H. Witten and J.G. Cleary, 1989. Modelling for text compression. ACM Comput. Surveys, 21: 557-597.

Bloom, C.R., 1996. LZP: A new data compression algorithm. Proceedings of the Conference on Data Compression, March 31-April 3, IEEE Computer Society, Washington, DC., USA., pp: 425-425.

Burrows, M. and D.J. Wheeler, 1994. A block-sorting lossless data compression algorithm. Digital Equipment Corporation (SRC Research Report 124).

Cleary, J.G. and I.H. Witten, 1984. Data compression using adaptive coding and partial string matching. IEEE Trans. Commun., 32: 396-402.

Deorowicz, S., 2003. Universal lossless data compression algorithms. Ph.D. Thesis, Silesian University of Technology.

Deutsch, L.P., 1996. RFC1951: DEFLATE Compressed Data Format Specification. Version 1.3, RFC Editor, USA.

Fauzia, S.A. and A. Mukherjee, 2001. LIPT: A lossless text transform to improve compression. Proceedings of the International Conference on Information Technology: Coding and Computing, April 2-4, IEEE Xplore Press, Las Vegas, NV., USA., pp: 452-460.

Hoang, D.T., P.M. Long and J.S. Vitter, 1999. Dictionary selection using partial matching. Inform. Sci., 119: 57-72.

Moffat, A. and R.Y.K. Isal, 2005. Word-based text compression using the Burrows-wheeler transform. Inform. Process. Manage., 41: 1175-1192.

Moffat, A., R.M. Neal and I.H. Witten, 1995. Arithmetic coding revisited. ACM Trans. Inform. Syst., 16: 256-294.

Nakamura, R., S. Inenaga, H. Bannai, T. Funamoto, M. Takeda and A. Shinohara, 2009. Linear-time text compression by longest-first substitution. Algorithms, 2: 1429-1448.

Nakano, Y., H. Yahagi, Y. Okada and S. Yoshida, 1994. Highly efficient universal coding with classifying to sub dictionaries for text compression. Proceedings of the Conference on Data Compression, March 29-31, Snowbird, UT., pp: 234-243.

Nelson, M. and J.L. Gailly, 1995. The Data Compression Book. 2nd Edn., M and T Books, New York, ISBN: 1-55851-434-1, pp: 541.

Robert, L. and N. Nadarajan, 2009. Simple lossless preprocessing algorithms for text compression. IET Software, 3: 37-45.

Say, B., D. Zeyrek, K. Oflazer and U. Ozge, 2002. Development of a corpus and a treebank for present-day written Turkish. Proceedings of the 11th International Conference of Turkish Linguistics, (ICTL'02), Eastern Mediterranean University, Northern Cyprus, pp: 183-192.

Shkarin, D., 2002. PPM: One step to practicality. Proceedings of the Data Compression Conference, April 2-4, IEEE Computer Society, Washington, DC., USA., pp: 202-211.

Skibiński, P., S. Grabowski and S. Deorowicz, 2005. Revisiting dictionary-based compression. Software- Practice Experience, 35: 1455-1476.

Skibiński, P. and J. Swacha, 2009. The efficient storage of text documents in digital libraries. Inform. Technol. Lib., 28: 143-153.

Storer, J.A. and T.G. Szymanski, 1982. Data compression via textual substitution. J. ACM, 29: 928-951.

Teahan, W. and J. Cleary, 1997. Models of English text. Proceedings of the Conference on Data Compression, (CDC'97), Snowbird, UT., USA., pp: 12-21.

Welch, T.A., 1984. A technique for high-performance data compression. Computer, 17: 8-19.

Williams, R.N., 1991. An extremely fast Ziv-lempel data compression algorithm. Proceedings of the Conference on Data Compression, (CDC'91), Snowbird, UT., USA., pp: 362-371.

Ziv, J. and A. Lempel, 1977. A universal algorithm for sequential data compression. IEEE Trans. Inform. Theory, 23: 337-343.

Ziv, J. and A. Lempel, 1978. Compression of Individual sequences via variable-rate coding. IEEE Trans. Inform. Theory, 24: 530-536.