Dictionary Based Text Filter for Lossless Text Compression

Rexline S. J [#1], Robert L [*2], Trujilla Lobo.F [#3]

[#]Department of Computer Science, Loyola College, Chennai, India
[*]Department of Computer Science Government Arts College,Coimbatore,India
[#]Department of Computer Science, Loyola College, Chennai, India

Abstract —This paper presents a new text transformation technique called Dictionary Based Text Filter for Lossless Text Compression. A text transformation technique should preserve the data during the encoding and decoding process. In the proposed approach, words in the source file are replaced with shorter codewords, whenever they are present in an external static dictionary. The rapid advantage of text transformation is that codewords are shorter than actual words and, thus, the same amount of text will require less space. As we are aware, 16% of the characters in the text files are spaces on average and hence to achieve better improvement of the compression rates for text files, the space between words can be removed from the source files. The unused ASCII characters from 128 to 255 are used to generate the codewords. This codeword combination chosen helps us to remove the space between the words in the encoded file. The proposed algorithm has been implemented and tested using standard Corpuses and compresses the files up to 85% reduction of its source file. We recommend the use of this proposed technique to compress the large text files in the field of the digitalization of library.

Keywords — *Decoder, Encoder, text transformation, preprocessing, Text Filter.*

I. Introduction

Text compression is a process which reduces the size of the original file without any loss of information in which it saves storage space and reduces the communication costs and also it reduces the time taken to search the pattern or portion of a file through the compressed file [18]. Therefore, text compression is considered to be an important research area to improve its algorithms and compressing technologies. Lossless data compression techniques are often partitioned into statistical based compression techniques and dictionary based compression techniques. Statistical compression algorithm is based on the probability that certain character will occur. Huffman Coding [14] and Arithmetic Coding [24] are the kind of statistical coders.

Dictionary based compression method exploits repetitions in the data. This coding scheme makes use of the fact that certain groups of consecutive characters occur more than once and assign a codeword to that certain occurrences. Most of the dictionary coders are based on LZ77 and LZ78 and are widely employed to compress the data. The LZW is also one of the dictionary based compression algorithm in which the dictionary is created dynamically and index values are used to represent the repeated dictionary words [18]. The advantage of LZW over the LZ77-based algorithms is its speed because of the limited numbers of string comparisons is enough to perform [23].

Transformation is another one methodology used to improve the compression performance in lossless text compression area [1, 10]. Researchers have proved that word based transformation method saves the run time memory, attaining good compression rates and also speeds up the transmission time [22]. Though there exists different word based preprocessing methods, there is a possibility for a better word based Preprocessor as the days and technologies progress [10]. There are major methods available for text preprocessing algorithm like Star Encoding, Length-Preserving Transform (LPT), Reverse Length-Preserving Transform (RLPT), Shortened – Context Length-Preserving Transform (SCLPT) [10], and Length Index Preserving Transform (LIPT) [4]. Text preprocessing method consists of taking a sequence of characters or alphanumeric strings [13] and transforming them into codewords. For effective compression, the resultant sequences of codewords will be smaller than the original sequence of characters. It is enough to use maximum of three bytes as codeword for entire words in the file [12]. The transformation is reversible such that the original sequence of characters can be recovered with no loss of information.

A text preprocessing technique, which goes well with the help of existing compressors like bzip2 contribute better compression ratio [10, 17]. But this method needs an external dictionary to store the words and also the order in which words appear in the dictionary has an impact in compression [22]. So that we need to be aware of high runtime memory requirement and more time consuming which are essential to care for this method in a healthy way.

The paper is organized as follows: Section II presents the existing text transformation techniques;

Section III proposes our new approach. Section IV substantiates the achievability and competence of the proposed method and finally Section V contains the conclusions.

II. Related Works

As we are aware, the text transformation techniques boost the compression rates up to few percent. The text transformation is a process, which reversibly transforms a data into some intermediate form. This transformed data then can be compressed with most of existing lossless data compression algorithms, like bzip2, gzip with better compression efficiency than achieved using an untransformed data. The reverse process is that decompression using given compressor like bzip2, gzip and a reverse preprocessing transformation.

The Burrows–Wheeler Transform (BWT) [5,8] is one of the best transformation methods in lossless text compression research area. BWT is a reversible transform that transfers the data into intermediate format that is generally more compressible. The Burrows - Wheeler Transform (BWT) encodes a block of data separately as a single unit. Even though, BWT constructs the transformed data larger than its original form, but the transformed data is more compressible than the untransformed data. This algorithm was developed by David J. Wheeler in 1983. It was on hand widely by Michael Burrows and David J. Wheeler in 1994 as a part of block-sorting compression algorithm. Bzip2 compressing algorithm compresses files using the Burrows-Wheeler block sorting text compression algorithm, and Huffman coding. Bzip2 compresses large files in blocks. The block size affects both the compression ratio attained and the amount of memory needed for compression and decompression. The internal algorithms used in BWT like Move-To-Front, arithmetic coder are modified to improve the performance of BWT [8]. Chapin [6,7] describes the method of reordering the alphabets instead of using lexicographic sorting in BWT which gives better improvements in BWT Algorithm. Recently,BWT has initiated many applications to bioinformatics field too.

R. Franceschini and A. Mukherjee proposed the Star Encoding [10] algorithm to transform the data into some intermediate form, which can be compressed with better efficiency. The star encoding makes use of * called signature of the character to substitute certain characters in a word and maintain few characters so that the word is still reversible. This star Encoding method attained better gain in compression rates. And also different preprocessing Schemes like Length-Preserving Transform (LPT) in which words of length more than four are encoded, Reverse Length-Preserving Transform (RLPT)) which is a revision of LPT, Shortened –Context Length-Preserving Transform (SCLPT) are designed to get better compression ratio. In which SCLPT outperforms other transforms and achieves better compression rates [10]. All these transformation method uses the static dictionary both in encoding and decoding process. Encrypted word based dictionary is also designed and tested which produced better results compared with un-encrypted word based dictionary [20].

The Star Encoding is improved and proposed by F .Awan and A. Mukherjee called as Length Index Preserving Transform (LIPT) [4] ,in which the codeword format is that of the symbol * followed by the word length indicated by the alphabets[a-z , A-Z] and then the index to the sub dictionary. The symbol * is used to indicate the transformed word which distinguishes the original words and the codewords. LIPT also used the static dictionary of size 0.5 MB in uncompressed format.

V.K. Govindan and B.S. Shajee mohan [11] proposed that the actual codeword consists of the length of the code concatenated with the code and the codewords are created using the ASCII characters 33 to 250. ASCII characters 251 to 254 used to represent the length of the code .A flag (ASCII 255) is used to indicate the absence of a space. If the character is one of the ASCII characters 251-255, the character is placed twice so as to show that it is part of the text and not a flag.

Md. Ziaul Karim Zia, Dewan Md. Fayzur Rahman, and Chowdhury Mofizur Rahman[25] describes a new transformation that the code words are generated using the ASCII characters (33 -128). Spaces between words and unused bit of ASCII character representation from each character are recovered to save one byte per 8 ASCII characters. Weifeng Sun,Amar Mukherjee and Nan Zhang suggested StarNT[22] method by introducing ternary search tree for encoding process and hashing to speed up the decoding process and used the alphabets [a-z,A-Z] for codeword .Grabowski [21] extended the StarNT transformation with several different algorithm like Space stuffing around the words, EOL coding, Binary filtering technique, Capital conversion, n-gram replacement to improve the preprocessing techniques. His preprocessing algorithm proposed hashing method to speed up the transformation.

Joaquin Adiego, Miguel A. Martinez-Prieto, Pablo de la Fuente [15] proposed a semi-static word-codeword mapping method that takes advantage of previous knowledge of some statistical data of the vocabulary which also retains all the desirable features of a word-codeword mapping technique. Miguel A. Martinez-Prieto, Joaquin Adiego, Pablo de la Fuente [19] presented Edge-Guided (E-G), an optimized text preprocessing technique which transforms the original text into a word net, which stores all relationships between adjoining words and proved that the best results are achieved when E-G preprocessing is coupled with high-order compressors such as Prediction by Partial Matching

(PPM) and also they stated that the best option is to use a word based PPM in conjunction with the spaceless word concept[16].

Antonio Farina, Gonzalo Navarro, Jose R. Parama [3] proposed Semistatic word-based byte-oriented compressors with new suffix-free Dense-Code-based compressor and proved that it achieves much better space , time and compression performance . R. R. Baruah, V.Deka , M. P. Bhuyan.[26] discussed about the preprocessing method to get better compression ratio.

III. PROPOSED TEXTFILTER

In this section, we propose our new techniques in text preprocessing method. A static dictionary is used to store the frequently used English words. Short codewords are assigned to the words present in the dictionary to do some precompression in the preprocessing stage itself. The size of the dictionary is limited according to the number of codeword available. This Text Filter method takes advantage of repetitions in the data and so it achieves better context to the existing compressors.

Each codeword should be unique. Codewords can be formed by two to three of the ASCII characters from 128 to 255, since they are never used in text files. Two length codeword cycles through the ASCII characters [128 - 245]. Totally, 13,924 words can be assigned two letter codeword. Three length codewords start with the ASCII character [246-255] and the next two characters cycles through the ASCII characters [128 - 245]. In this way, 10 x 118 x 118 = 139,240 three letter codeword can be used in our transformation. Totally, maximum of 153,164 codewords can be assigned for the words in the static dictionary and the number of codewords possible according to the above combination is more than enough for the very often repeated words in English language [22].

The reason why we have chosen the unused ASCII characters to generate codeword and also the above said combination of codeword is that this kind of codeword combination helps us to remove the space between the words. Removing the spaces form the source file is a well-known transformation in such word-based text compression that improves the compression ratio [9]. Since we have planned to remove the space between the words, we analyzed the spaces between the words in the sample files from the Calgary corpus and Canterbury Corpus. Table I shows the space frequencies of the files as known from Abel J and Teahan W [2]. By using this approach, we recover 16% of space on average from any text file.

The space frequency results, mentioned in Table I of this paper, provided a sturdy origin to make improvement in the text filtering method. There are no codeword for digits, punctuation, tab and EOL. So that the digits, punctuation and EOL are transferred as it is. F. Awan and A. Mukherjee [4, 21] proved that capital conversion is a recognized preprocessing technique.

Words started with capital letter are converted to their lowercase equivalent and full uppercase words are also converted to its lowercase form and indicated the changes with a flag. Those words that are not present in the dictionary are not converted to codeword and it is transferred as it is and indicated with a flag ‘ASCII character 127’. This flag is used to recover the space between the unaltered words present consecutively while decoding the text.

TABLE I. SPACE FREQUENCIES OF CALGARY AND CANTERBURY CORPUS

File Names	File Size (Bytes)	Spaces (Bytes)	Spaces %
Bib	111261	13739	12.35
book1	768771	125551	16.33
book2	610856	556047	14.06
News	377109	54269	14.39
paper1	53161	7301	13.73
paper2	82199	12112	14.73
Progc	39611	6925	17.48
Progl	71646	12238	17.08
Progp	49379	11474	23.24
Trans	93695	9901	10.57
Bible	4047392	766111	18.93
World192	2473400	428662	17.33
Average			15.85

The Text Filter algorithm we developed is a three step process consisting of Dictionary Creation Algorithm, Encoding Algorithm and Decoding Algorithm. The procedure used in the proposed method can be summarized as follows.

A. *Dictionary Creation Algorithm*

Using multiple source files as input, a Static Dictionary can be created with fixed sequences of words by extracting all words from input files, sorting the Dictionary by frequency of occurrences of the word in descending order and the lower case versions of the words are stored in the dictionary. Then codeword are assigned for the words in the dictionary using the ASCII characters 128 to 255. Two length codeword cycles through the ASCII characters [128 - 245]. Totally, 13,924 words can be assigned two letter codeword. Three length codeword starts with the ASCII character [246-255] and the next two characters cycles through the ASCII characters [128 - 245]. In this way, 10 x 118 x 118 = 139,240 three letter codeword can be used in our transformation. According to the mapping mechanism, it is possible to store up to 118 x 118 + 10 x 118 x 118 = 153,164 different two and three letter codewords in the Dictionary.

B. Encoding Algorithm

The words in the source file are searched in the Dictionary. If the input text word is found in the dictionary, replace the word with the codeword assigned. If the input word is not found in dictionary, then it is transferred as it is by placing a flag 'ASCII value 127' before the unaltered word. This flag helps us to recover the space between the consecutive unaltered words present in the encoded file. Words that start with capital letter are converted to their lowercase equivalent and to denote this change, the 'ASCII value 12' is used as a flag and placed in front of the respective codeword. Moreover it is worth using another 'ASCII value 11' as a flag to mark a conversion of a full uppercase word to its lowercase form to indicate that change. If all the letters of a word are in lower case then no flag is placed before their codeword. Space between words is removed without transformed into the intermediate file. If more than one space is there between the words, except only one space, all other spaces are encoded into the transformed file as it is. Punctuation, EOL, and digits are not converted to codeword and transferred as it is. If the input character is the character used for codeword and flag, then another flag 'ASCII value 6' is placed before that just to indicate that the character is part of the source file and not a codeword or flag. Once all the input text has been transformed according to the above steps, then the transformed text is fed to the existing backend compressors like Bzip2, PPM, gzip etc.

C. Decoding Algorithm

The compressed text is first decompressed using the same compressor as it was compressed at the source end and the transformed text is recovered. The reverse transformation is applied on this decompressed transformed text. If the codeword starts with the ASCII character 128 to 245, then consider only the consecutive two characters as a codeword and find for the matching the word in the dictionary. If the codeword starts with the ASCII character 246 to 255, then consider only the consecutive three characters as a codeword and find for the matching the word in the dictionary. The transformed codewords are replaced with the respective words in the dictionary. The unaltered word can be easily recoganzied by the flag (ASCII value 127) and transformed as it is in the decoded file by stuffing a space in the decoded file . Between each and every codeword, space should be inserted to recover the source file. The change of capitalization of the word is also performed using the respective ASCII flag identifier.

IV. PERFORMANCE ANALYSIS

When we focus our attention to evaluate the performance and excellence of good compression algorithm, there are several criteria to be under consideration such as, the compression ratio, memory requirements and timing performance in the case of lossless text compression. In this section, we compare the performance of our proposed Dictionary Based Text Filter with the backend algorithm Bzip2. The reason to use bzip2 as our backend compressor is that bzip2 compresses files using the Burrows-Wheeler block sorting text compression algorithm and Huffman coding and also bzip2 outperforms other compression algorithms when compared with Gzip, Gzip-9, and DMC by giving the best compression ratios with lowest execution time [10]. Our experiments were carried out on an 800MHz equipped with 3.00 GB RAM, under Windows Vista operating system. Dictionary Based Text Filter was implemented in VC++ (Microsoft Visual Studio 2005).

A. Compression Ratio of Dictionary Based TextFilter

The compression ratios are expressed in terms of average BPC (bits per characters). We compared the compression performance of our proposed Dictionary Based Text Filter with the results of Bzip2 and LIPT as listed on Table II and it can be seen that our transform algorithm outperforms almost all the other improvements. The detailed compression results in terms of BPC for our test corpus are summarized as follows.

TABLE II. COMPRESSION RATIO (BPC) OF TEXTFILTER WITH BZIP2 AND LIPT WITH BZIP2

File Names	File size Bytes	Bzip2 (BPC)	LIPT+ Bzip2 (BPC)	Bzip2+ Text Filter (BPC)	File Names	File size Bytes	Bzip2 (BPC)	LIPT+ Bzip2 (BPC)	Bzip2+ Text Filter (BPC)
CALGARY CORPUS					*CANTERBURY CORPUS*				
Bib	111261	1.98	1.93	1.71	Grammer. lsp	3721	2.76	2.58	2.71
book1	768771	2.42	2.31	2.26	xargs.1	4227	3.33	3.10	2.82
book2	610856	2.06	1.99	1.93	fields.c	11150	2.18	2.14	1.92
News	377109	2.52	2.45	2.32	Cp.html	24603	2.48	2.44	2.71
paper1	53161	2.49	2.33	2.25	asyoulik	125179	2.53	2.42	2.28
paper2	82199	2.44	2.26	2.15	alice29	152089	2.27	2.13	2.05
Paper3	46526	2.72	2.45	2.35	lcet10	426754	2.02	1.91	1.78
Paper4	13286	3.12	2.74	2.6	plrabn12	481861	2.42	2.33	2.25
Paper5	11954	3.24	2.95	2.79	world192	2473400	1.58	1.52	1.37
Paper6	38105	2.58	2.40	2.3	bible	4047392	1.67	1.62	1.57
Progc	39611	2.53	2.44	2.34	Average (BPC)		2.32	2.22	2.15
Progl	71646	1.74	1.66	1.63	*GUTENBERG CORPUS*				
Progp	49379	1.74	1.72	1.62	1musk10	1344739	2.08	1.98	1.88
Trans	93695	1.53	1.47	1.42	World95	2988578	1.54	1.49	1.40
Average (BPC)		2.36	2.22	2.12	Anne	586960	2.22	2.12	2.03
					Average (BPC)		1.95	1.86	1.77

The average BPC using original Bzip2 is 2.36 and Bzip2 with LIPT gives average BPC of 2.22, according to experimental results based on the Calgary Corpus. According to Canterbury files based results, the average BPC using original Bzip2 is 2.32 and Bzip2 with LIPT gives average BPC of 2.22. According to Gutenberg files based results taken from F. Awan and A. Mukherjee, [4] , the average BPC using original Bzip2 is 1.95 and Bzip2 with LIPT gives average BPC of 1.86 .But based on our method, average BPC for Calgary Corpus is 2.12, a 10.17**%** improvement over bzip2 and 4.5% improvement over LIPT and for Canterbury files is 2.15 BPC, a 7.33**%** improvement over bzip2 and 3.15% improvement over LIPT and average BPC for Gutenberg files is 1.77, a 9.23**%** improvement over bzip2 and 4.84% improvement over LIPT. Table III shows the compression performance of Textfilter over PPMd which is a representative of the PPM family and the p7zip compressor which is a LZMA algorithm based compressor. It shows clearly that Textfilter combined with PPMd and 7Z outperforms in compression ratio.

Table IV shows the compression ratios achieved in our tests. A “None” means that the existing compressor like Bzip2, PPMd and 7Zip was applied over plain text. We compared our results with End-Tagged Dense Code (ETDC) and (s, c)-Dense Code (SCDC) used as a preprocessing step with the backend compressor taken from Antonio Farina, Gonzalo Navarro, Jose R. Parama[3]. We used the text files of the Calgary corpus collection: book1, book2, bib, news, and paper1-6 for comparison.It can be seen that dense+bzip2 improves bzip2 and Dense+p7zip overcomes p7zip dense+PPMd works similarly to MPPM [3].But according to the results shown in Table IV, our proposed Textfilter outperforms all the other methods.

TABLE III. COMPRESSION RATIO (BPC) OF TEXTFILTER WITH PPMD AND 7Z

File Names	PPMd (Bytes)	PPMd (BPC)	PPMd+ Text Filter (Bytes)	PPMd +Text Filter (BPC)	7Z (Bytes)	7Z (BPC)	7Z +TextFilter (Bytes)	7Z +TextFilter (BPC)
Bib	25479	1.83	23392	1.68	30716	2.21	26380	1.90
book1	215845	2.25	202270	2.10	261064	2.72	231061	2.40
book2	149437	1.96	139010	1.82	169894	2.22	152923	2.00
News	109552	2.32	102001	2.16	119399	2.53	109416	2.32
paper1	14999	2.26	13835	2.08	17322	2.61	14981	2.25
paper2	22881	2.23	20803	2.02	27310	2.66	23022	2.24
Paper3	14432	2.48	13061	2.25	17097	2.94	14210	2.44
Paper4	4623	2.78	3906	2.35	5444	3.28	4273	2.57
Paper5	4298	2.88	3885	2.60	4938	3.30	4183	2.80
Paper6	11104	2.33	10160	2.13	12552	2.64	10843	2.28
Progc	11344	2.29	10624	2.15	12605	2.55	11508	2.32
Progl	14844	1.66	13751	1.54	15060	1.68	14124	1.58
Progp	10240	1.66	9406	1.52	10428	1.69	9897	1.60
Trans	17104	1.46	15146	1.29	16896	1.44	15743	1.34
Average (BPC)		2.17		1.98		2.46		2.15

B. *Memory Space Complexity*

In our implementation, the transform dictionary is a static dictionary shared by both encoding process and decoding process. The drawback of Dictionary Based Text Filter is the requirement of maintaining relatively a large collection of words as a static dictionary. Such a dictionary may contain frequently occurring words of arbitrary length, digrams or n-grams. This kind of dictionary can easily be built upon an existing coding such as ASCII value 128 to 255 by using previously unused codewords or extending the length of the codewords to accommodate the dictionary entries. Another measure that affects the transformation is that, the number of words in the dictionary and the frequencies of occurrence of each word in the source file are greatly affects the compression rates.

In our method, we tested our dictionary by placing the words up to 130,000. Even though, we limited the dictionary size, it could be possible to place up to 150,000 words based on the requirement of the source file. A significant way to minimize the memory usage and storage space is to reduce the number of words in the dictionary. Moreover, the best way of shortening the size of the dictionary is to avoid the scarcely used words. We tested the implementation with dictionary of size 1.67MB uncompressed and 595KB when compressed with Bzip2. Dictionaries would have to be transmitted only once, and could be reused [6]. Another one solution to handle the dictionary is that the created dictionary could be uploaded to a public directory on a web site.

TABLE IV. COMPARISION ON COMPRESSION RATIO OF TEXTFILTER WITH ETDC AND SCDC

Preprocessor	PPMd	7Z	Bzip2
CALGARY CORPUS			
None	25.36%	29.96%	28.92%
ETDC	20.04%	29.38%	31.67%
SCDC	27.97%	29.48%	31.23%
Textfilter	25.19%	27.98%	26.87%

C. *Timing Performance*

In any compression process, time complexity of the transformation method plays a vital role. In LIPT, binary search technique is used to expedite searching [4]. But it is proved that hashing techniques used for searching the words in the dictionary speeds up the encoding and decoding time taken by the preprocessor[21,22] with less memory. We used hashing method for dictionary mapping in our implementation to expedite for dictionary mapping. Moreover, as we are aware that the timing performance of the preprocessor is machine dependent one. Transformed text compresses with better efficiency over existing compressors like Bzip2, Gzip [4]. Our algorithm also gives better improvement in transmission time over Bzip2.

V. CONCLUSIONS

Our proposed TextFilter method is admirable extensions of the transformation mechanisms which considerably reduces the disk space required to store the text files experimented on various texts Corpuses. This transformation algorithm also includes an efficient dictionary mapping mechanism to remove the space between the words in the source files to recover substantial amount of memory. It is very clear that this new transformation technique with bzip2 could provide a better compression performance of upto 85 % reduction in size of source file and also it maintains an appealing compression and decompression speed. Since the dictionary is a static one and due to the increasing interest in development of digital library related technologies, we have prior knowledge about the content of the source file, thus different dictionaries can be created and invoked which gives the most significant improvement in the compression performance.

REFERENCES

[1] Abel J. "*Record preprocessing for data compression*", Proceedings of the 2004 IEEE Data Compression Conference, IEEE Computer Society Press, Los Alamitos, California, pp .521,2004.

[2] Abel,J, Teahan,W, "Universal Text Preprocessing for Data Compression",IEEE Trans.Computers,54(5)pp :497-507,2005.

[3] Antonio Farina, Gonzalo Navarro, Jose R. Parama, "Boosting Text Compression with Word-Based Statistical Encoding." The Computer Journal, 55(1): 111-131 (2012).

[4] F. Awan and A. Mukherjee, "LIPT: A Lossless Text Transform to Improve Compression," Proceedings of International Conference on Information and Theory:Coding and Computing, IEEE Computer Society, pp. 452-460, April 2001.

[5] M. Burrows and D.J. Wheeler, "A Block-Sorting Lossless Data Compression Algorithm", SRC Research Report 124, Digital Systems Research Center, Palo Alto, CA, 1994.

[6] Chapin, B. "Higher Compression from the Burrows-Wheeler Transform with new Algorithms for the List Update Problem", Ph.D. Dissertation, University of North Texas, 2001.

[7] Chapin B, Tate SR."Higher Compression from the Burrows–Wheeler Transform by Modified Sorting", In Storer JA, Cohn M, editors, Proceedings of the 1998 IEEE Data Compression Conference, IEEE Computer Society Press, Los Alamitos, California,pp.532,1998.

[8] S.Deorowicz,"Improvements to Burrows-Wheeler Compression Algorithm ", Software – Practice and experience, pp.1465-1483, 2000.

[9] Edleno de Moura, Gonzalo Navarro, Nivio Ziviani: "Indexing Compressed Text". WSP'97: 95-111.

[10] R. Franceschini, H. Kruse, N. Zhang, R. Iqbal, and A. Mukherjee, "Lossless, Reversible Transformations that Improve Text Compression Ratio," Project paper, University of Central Florida, USA. 2000.

[11] V.K. Govindan, B.S. Shajee mohan, "IDBE – An Intelligent Dictionary Based Encoding Algorithm for Text Data Compression for High Speed Data Transmission Over Internet", Proceeding of the International Conference on Intelligent Signal Processing and Robotics IIIT Allahabad February 2004.

[12] H.S. Heaps. "Information Retrieval - Computational and Theoretical Aspects". Academic Press, 1978.

[13] Horspool N, Cormack G. "Constructing Word-Based Text Compression Algorithms", Proceedings of the 1992 IEEE Data Compression Conference, IEEE Computer Society Press, Los Alamitos, California, pp. 62–71,1992.

[14] Huffman, D.A.," A method for the construction of minimum-redundancy codes". Proc. Inst. Radio Eng., 40: pp: 1098-1101.1952.

[15] Joaquin Adiego, Miguel A. Martinez-Prieto, Pablo de la Fuente: "High Performance Word-Codeword Mapping Algorithm on PPM". DCC 2009: 23-32

[16] Joaquin Adiego, Pablo de la Fuente: "Mapping Words into Codewords on PPM". SPIRE 2006: 181-192.

[17] H. Kruse and A. Mukherjee, "Preprocessing Text to Improve Compression Ratios", Proceedings of Data Compression Conference, IEEE Computer Society, Snowbird Utah, pp. 556, 1998.

[18] U. Manger, "A Text compression scheme that allows fast searching directly in compressed file" , ACM Transactions on Information Systems, Vol.52, N0.1, pp.124-136, 1997.

[19] Miguel A. Martinez-Prieto, Joaquin Adiego, Pablo de la Fuente: "Natural Language Compression on Edge-Guided text preprocessing. Information Sciences", 181(24): 5387-5411 (2011)

[20] Robert Franceschini, Amar Mukherjee, " Data Compression Using Encrypted Text ",proceedings of the third forum on Research and Technology, Advances on Digital Libraries,ADL 96,pp .130-138, May 1996.

[21] P. Skibiński, Sz. Grabowski and S. Deorowicz. "Revisiting dictionary-based compression". Software–Practice and Experience, pp.1455-1476, 2005.

[22] Sun W, Mukherjee A, Zhang N. "A Dictionary-based Multi-Corpora Text Compression System" . In Storer JA, Cohn M, editors, Proceedings of the 2003 IEEE Data Compression Conference, IEEE Computer Society Press, Los Alamitos, California, pp .448 ,2003.

[23] Weiling Chang, Binxing Fang, Xiaochun Yun, Shupeng Wang" The Block Lossless Data Compression Algorithm", International Journal of Computer Science and Network Security, VOL.9 No.10, October 2009.

[24] Witten, I.H., R.M. Neal and J.G. Cleary, "Arithmetic coding for data compression",. Commun.ACM, 30: pp : 520-540.,1987

[25] Md. Ziaul Karim Zia, Dewan Md. Fayzur Rahman, and Chowdhury Mofizur Rahman, "Two-Level Dictionary-Based Text Compression Scheme", Proceedings of 11th International Conference on Computer and Information Technology, Khulna,Bangladesh.,pp.25-27,December-2008.

[26] R. R. Baruah , V.Deka , M. P. Bhuyan. "Enhancing Dictionary Based Preprocessing for Better Text Compression ". International Journal of Computer Trends and Technology (IJCTT) V9(1):4-9, March 2014.