RIDBE: A Lossless, Reversible Text Transformation Scheme for better Compression

S. Senthil
Department of Computer Science,
Vidyasagar College of
Arts and Science,
Udumalpet, Tamilnadu,
India

S. J. Rexiline
Department of Computer Science,
Loyola College,
Chennai,
Tamilnadu,
India

L. Robert
Computer Science & Info. System
Department, Community College in
Al-Qwaiya, Shaqra University, KSA
(Government Arts College,
Coimbatore)

ABSTRACT

In this paper, we propose RIDBE (Reinforced Intelligent Dictionary Based Encoding), a Dictionary-based reversible lossless text transformation algorithm. The basic philosophy of our secure compression is to preprocess the text and transform it into some intermediate form which can be compressed with better efficiency and which exploits the natural redundancy of the language in making the transformation. In RIDBE, the length of the input word is denoted by the ASCII characters 232 – 253 and the offset of the words in the dictionary is denoted with the alphabets A-Z. The existing or backend algorithm's ability to compress is seen to improve considerably when this approach is applied to source text and it is used in conjunction with BWT. A sufficient level of security of the transmitted information is also maintained. RIDBE achieves better compression at the preprocessing stage and enough redundancy is retained for the compression algorithms to get better results. The experimental results of this compression method are analysed. RIDBE gives 19.08% improvement over Simple BWT, 9.40% improvement over BWT with *-encode, 3.20% improvement over BWT with IDBE, 1.85% over BWT with EIDBE and about 1% over IIDBE.

Keywords

Compression, Decompression, Preprocessing, Dictionary methods

1. INTRODUCTION

It has been established through empirical research, which believes in experiments rather than theories, that compression algorithms can play a crucially central role in reducing redundancy in data representation. The storage required for data representation is decreased sizably by compression algorithms. Another fruitful finding of the research is that communication costs can be reduced considerably through able and judicious administration of available bandwidth by data compression. Research in the last ten years has triggered an unprecedented explosion in the volume of digital data transmitted over the Internet, representing text, images, video, sound, computer programs etc. An era of even greater digital data explosion can be envisioned if the momentum of research is maintained in the same vein, and then, hopefully, much improved algorithms, born of relentless research, will result in a remarkably maximal compression of data through effective use of available network bandwidth.

Developing different compression algorithms is envisaged as one approach for attempting to attain better compression ratios. Of the number of sophisticated algorithms already available for lossless text compression , Burrows Wheeler Transform (BWT) [5] and Prediction by Partial Matching [14] prove to be much superior to the classical algorithms like Huffman, Arithmetic and LZ families [25] of Gzip and Unix – compress [24] in performance. While the ratio of compression achieved by PPM is much higher than that of almost all existing compression algorithms the main handicap of PPM is that it is very slow and also consumes large amount of memory to store context information. The cyclic rotations of a block of data generating a list of every character and its arbitrarily long forward context are lexicographically sorted by BWT. It makes use of Move-To-Front (MTF) [1] and an entropy coder as the backend compressor. Unceasing efforts are on to improve the efficiency of PPM [8,9,17] and BWT [1,3,19].

The main focus of this paper relates to an alternative approach which seeks to perform a lossless, reversible transformation to a source file before applying an existing compression algorithm. The compression of the source file in a much easier way is accomplished through the said transformation. The input to the transformation is the source file whereas the transformed text is the output. The output is fed to an existing data compression algorithm and the outcome is an efficient compression of the transformed text. A reversal of this process is required to be done for decompression by first invoking the appropriate decompression algorithm, and then providing the resulting text to the inverse transform. The preservation of the overall lossless text compression paradigm without any compromise is totally dependent upon the exactly reversible transformations. The data compression and decompression algorithms are unmodified, so they do not exploit information about the transformation while compressing.

Figure 1 illustrates the paradigm.

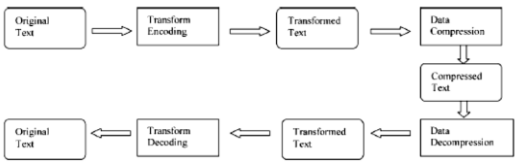

Fig 1: Text compression paradigm incorporating a lossless, reversible transformation.

The paradigm is used with the singular objective of enhancing the compression ratio of the text in comparison with what could have been achieved by using only the compression algorithm. It has been observed that the preprocessing of the text prior to conventional compression will improve the compression efficiency much better.

The preprocessing of textual data is a subject of many publications. In some articles, the treatment of textual data is embedded within the compression scheme itself but can easily be separated into two independent parts: a preprocessing algorithm and a standard compression algorithm, which are processed sequentially one after the other.

It is of utmost importance that the security aspects of the data being transmitted while compressing it should be given top priority as there is always an impending threat of the text data transmitted over the internet being subjected to a host of hostile attacks.

Our endeavour in this paper is to experiment and develop a more efficient transformation yielding greater compression to the text data. Our encoding method "Reinforced Intelligent Dictionary Based Encoding (RIDBE)" is used as a preprocessing stage so as to improve the compression ratio.

2. REVIEW OF LITERATURE

Burrows and Wheeler [5] describes a block sorting lossless data compression known as Burrows Wheeler Transform (BWT) that takes a block of data and reorders it using a sorting algorithm. The resulting block of text contains the same symbols as the original, but in a different order. The transformation groups similar symbols, so the probability of finding a character close to another instance of the same character increases substantially. The resulting text can be easily compressed with fast locally adaptive algorithms, such as Move-to-Front coding combined with Huffman or arithmetic coding preceded by a Run – Length Encoding (RLE).

Kruse and Mukherjee [13,10] propose a special case of word encoding known as star encoding. This encoding method replaces words by a symbol sequence that mostly consist of repetitions of the single symbol '*'. This requires the use of an external dictionary that must be known to the receiver as well as the sender. Inside the dictionary, the words are first sorted by their length and second by their frequency in the English language using information obtained from Horspool and Cormack [11]. The requirement of an external dictionary makes this method again language dependent. The transformed text can now be the input to any available lossless text compression algorithm, including Bzip2 where the text undergoes two transformation, first the *-transform and then a BWT transform.

Awan and Mukherjee [2] describe a Dictionary based reversible, lossless text transformation called as Length Index Preserving Transform (LIPT) which can be applied to a source text to improve the ability to compress the existing algorithm. LIPT encodes a word as a string that can be interpreted as an index into a dictionary. BZIP2 with LIPT gives 5.24% improvement in average BPC over BZIP2 without LIPT and PPMD with LIPT gives 4.46% improvement in average BPC over PPMD without LIPT for test corpus.

Robert and Nadarajan [16] present different reversible preprocessing algorithms. In their paper they have described about Dictionary Based Transformation (DBT) and Dynamic Reversible Transformation (DRT). DBT reduces the total number of possible byte values used in a text file. When DBT is combined with Huffman an average of 1.2 BPC is saved. At the same time when it is combined with Arithmetic coding an average of 1.17 BPC is saved and when it is combined with LZW an average of 0.58 BPC is saved. When DRT is combined with Huffman reduction in BPC is observed. A significant saving in BPC is noted, when it is combined with Arithmetic coding. On the contrary, combination with LZW does not provide best BPC for some of the test files because LZW is an adaptive higher order data compression algorithm.

Chapin and Tate [6] and later Chapin [7] present preprocessing methods, specialized for a specific compression scheme. They describe several methods for alphabet reordering prior to using the BWCA in order to place letters with similar contexts close to one another. Since the Burrows-Wheeler transformation (BWT) is a permutation of the input symbols based on a lexicographic sorting of the suffices, this reordering places areas of similar contexts at the BWT output stage closer together, and these can be exploited by the latter stages of the BWCA. The paper compares several heuristic and computed reorderings where the heuristic approaches always achieve a better result on text files than the computed approaches. The average gain for BWCA using heuristic reorderings over the normal alphabetic order was 0.4% on the text files of the Calgary Corpus. Balkenhol and Shtarkov use a very similar heuristic alphabet reordering for preprocessing with BWCA [4]. Kruse and Mukherjee [12] describe a different alphabet reordering for BWCA. It also describes a bigram encoding method and a word encoding method which is based on their star encoding.

Sun et al. [22] describe a dictionary based fast lossless text transform algorithm called as StarNT, which utilizes ternary search tree to expedite transform encoding. This new transform achieves improvement not only in compression performance but also in time complexity when compared with LIPT. Facilitated with StarNT, bzip2 and PPMD achieves a better compression performance in comparison to most of the other recent efforts based on PPM and BWT. Experimental results show that StarNT achieves an average improvement in compression ratio of 11.2% over bzip2-9, 16.4% over gzip-9 and 10.2% over PPMD.

Shajeemohan and Govindan [21] propose an encoding strategy called Intelligent Dictionary Based Encoding (IDBE) which offers higher compression ratios and rate of compression. According to them, IDBE involves two stages. The first is the creation of an intelligent dictionary and the next one is encoding the input text data. While creating a dictionary, words are extracted from the input files and for the fist 218 words ASCII characters 33-250 are assigned as the code. For the remaining words permutation of two of the ASCII characters in the range of 33-250 is assigned. For the left out words, if any, permutation of three of the ASCII characters for each word and if required permutation of four of the ASCII characters is assigned. During encoding, the length of the token is determined and the length is concatenated with the code and is represented by the ASCII characters 251-254 with 251 for a code of length 1, 252 for length 2 and so on. While decoding the length proves to be the end marker. A better compression is achieved by using IDBE as the preprocessing stage for the BWT based compressor. The time for transmission of files has also been reduced to a greater extent.

In [18] we described an algorithm called "Enhanced Intelligent Dictionary Based Encoding" (EIDBE) to preprocess textual documents in order to improve the compression ratio of different standard compression algorithms. The basic idea of this preprocessing algorithm is to replace words in the input text by a character encoding that represents a pointer to an entry in a static dictionary. This algorithm consists of two steps. First is the creation of

intelligent dictionary and the second is encoding the input data. Words are extracted from the input files and they are categorised as two letter, three letter and so on up to twenty two letter words. And the first 199 words in each segment have single ASCII character representation (33-231) as the code and a marker character. For the words remaining permutation of two of the ASCII characters in the range of 33-231 is assigned. For the left out words, if any, permutation of three of the ASCII characters for each word and if required permutation of four of the ASCII characters is assigned. During encoding, the length of the token is determined and the length is concatenated with the code and is represented by the ASCII characters 232 - 253 with 232 for two letter word, 233 for three letter word and so on. The calculation reveals that from a two letter word to a twenty two letter word, single ASCII character representation could be achieved for 4179 words, which is phenomenal compared to IDBE.

In [19], we described an algorithm called "Improved Intelligent Dictionary based Encoding" (IIDBE) which is a reversible transformation that makes the text better compressible by applying it to a source text that improve an existing, or backend, algorithm's ability to compress. This algorithm has two steps. First one is the creation of the dictionary and the second is encoding the input data. The significant change in the creation of dictionary from EIDBE is for the first 52 words, ASCII characters 65 – 90 and 97 – 122 are assigned as the code. However encoding process remains the same.

3. REINFORCED INTELLIGENT DICTIONARY BASED ENCODING (RIDBE)

It is strongly felt that a more efficient encoding strategy, offering higher compression ratios, rate of compression and maintaining confidentiality of the data sent over the channel by making use of the dictionary for encoding and decoding will prove to be a much better alternative to the existing one, by being more viable and useful.

The first stage of the preprocessing block of the proposed compression scheme consists of two operations.

- The first operation is to transform text into some intermediate form with Reinforced Intelligent Dictionary Based Encoding (RIDBE) scheme.
- The encoding of the transformed text can then be carried out with a BWT stage.

The preprocessed text is then piped through a Move-To-Front encoder stage, then a Run Length Encode stage, and finally an Entropy encoder, usually Arithmetic coding.

The present research of creation of RIDBE, a preprocessing algorithm, involves two important steps namely creating a dictionary and encoding the input data.

The process of dictionary creation includes extracting words from the file and sorting them according to the length of the word and frequency of occurrence. Finally a dictionary is created by assigning codes to the words.

The techniques adopted in encoding are reading characters to form tokens and checking the tokens for their length and assigning an ASCII character which in turn serves to be the marker character for decoding.

3.1 Dictionary Creation Algorithm

START

Create Dictionary with source files as input

1. Extract words from the input files one by one and check whether they are already available in the table. If they are already available, increment the number of occurrences by one, otherwise add it and set the number of occurrence to one.
2. Sort the table by the length of the words in the Ascending order (Two letter words, Three letter words and so on).
3. Again sort the table by frequency of occurrences in Descending order according to the length of the word.
4. Start assigning codes with the following method:
   - Assign the first 26 (Two letter) words the characters A – Z (ASCII characters 65 – 90) as the code.
   - Now assign each of the remaining words permutation of two of the characters in the range of A - Z taken in order.
   - If any words remain without being assigned characters, assign each of them permutation of three of the characters and finally, if required, permutation of four of the characters.
5. Repeat the above procedure for three letter words, four letter words and so on up to twenty two letter words because the maximum length of an English word is 22 [15].
6. The created file which consists of only words and their codes serves as the dictionary file.

STOP

3.2 Encoding Algorithm

Start encoding with input file

A. Read the Dictionary file

B. While input file is not empty

   1. Read the characters from the input file and form tokens.
   2. If the token is longer than one character, then

      i) Search for the token in the table

      ii) If it is found,

         a. Find the length of the token

         b. The actual code consists of the length concatenated with the code in the table and the length serves as the end marker for decoding and is represented by the ASCII characters 232 – 253 with 232 for two letter words, 233 for three letter words, … and 252 for twenty two letter words and 253 for words which are greater than twenty two letter words.

      Else

         a. If the character preceding the token is a space, a marker character (ASCII 254) is inserted to indicate the presence of a space and if it is not

a space then a marker character (ASCII 255) is added to indicate the absence of a space.

iii) Write the actual code into the output file.

iv) Read the next character and

- If it is a space followed by any alphanumeric character, ignore the space.
- If it is a space followed by any non-alphanumeric character, a marker character (ASCII 254) is inserted to represent the presence of a space and if it is not a space but any other character, a marker character (ASCII 255) to indicate the absence of a space and the characters are written into the output file till another space or an alphanumeric character is encountered.
- Go back to B.

Endif

Else

i) Write the one character token.

ii) Before writing it, check the character preceding the one character token. If it is a space, a marker character (ASCII 254) is added to indicate the presence of the space and if it is not a space, a marker character (ASCII 255) is added to represent the absence of the space.

iii) If the characters is one of the ASCII characters (232 – 255), write the character once more so as to represent that it is a part of the text and not a marker character.

Endif

End (While)

C. Stop

3.3 Decoding algorithm

The compressed text received is first decoded using the same compressor used at the sending end and the encoded text is recovered.

Start decoding with the encoded file

A. Read the dictionary file

B. While encoded file is not empty

[Marker Characters: ASCII value of the character is greater than or equal to 232 and less than or equal to 253]

1. Read the next character and

a. If the ASCII character is within the range of 65 – 90 then

- A word is formed till a marker character is encountered.
- The word length is calculated and the word is searched in the dictionary file in the respective length block and at the respective position in that block.

If it is found then

- The corresponding English word for the code is found and it is written to the output file.
- Before writing it, if the ASCII value of the character preceding the marker character is 255, a space is not inserted and if it is 254, a space is inserted.
- Go back to 1.

Else

- Write it into the output file as such.
- Before writing it, if the ASCII value of the character preceding the word / character is 255, a space is not inserted and if it is 254, a space is inserted.
- Go back to 1.

Endif

b. Else if the ASCII character is within the range of 1 – 64 and 91 – 231 then

- Write it into the output file as such.
- Before writing it, if the ASCII value of the character preceding the word / character is 254, a space is inserted and if it is 255, space is not inserted.
- Go back to 1.

c. Else

- If the ASCII value of the character preceding the word / character is 254, a space is inserted and if it is 255, a space is not inserted.
- If the ASCII value of the character and its preceding character is same (i.e., 232 – 255), it is a part of the text and not a marker character and it is represented only once and before writing it check the third character to the left of the character read, and if the ASCII value is 255, a space is not inserted and if it is 254, a space is inserted.
- Go back to 1.

Endif.

End [While]

C. Stop

4. EXPERIMENTAL RESULTS

This section has as its main thrust a comparison of the performance of RIDBE with six cases: Simple BWT, BWT with Star encoding, BWT with IDBE, BWT with EIDBE and BWT with IIDBE and BWT with our proposed algorithm RIDBE. The measurements are centered around compression results in terms of BPC (Bits Per Character). Benchmark files from Calgary and Canterbury Corpuses are invariably used to validate the performance of the compression scheme.

Performance of RIDBE in comparison with Simple BWT, BWT with Star encoding, BWT with IDBE, BWT with EIDBE and BWT with IIDBE in Calgary Corpus is shown in Table 1.

Table 1. BPC Comparison of RIDBE with Simple BWT, BWT with Star encoding, BWT With IDBE, BWT with EIDBE and BWT with IIDBE in Calgary Corpus

File Names	File size in bytes	Simple BWT	BWT With * Encode	BWT with IDBE	BWT with EIDBE	BWT with IIDBE	BWT with RIDBE
		BPC	***BPC***	***BPC***	***BPC***	***BPC***	***BPC***
bib	1,11,261	2.11	1.93	1.69	1.76	1.76	1.74
book1	7,68,771	2.85	2.74	2.36	2.53	2.47	2.45
book2	6,10,856	2.43	2.33	2.02	2.18	2.15	2.13
news	3,77,109	2.83	2.65	2.37	2.52	2.49	2.48
paper1	53,161	2.65	1.59	2.26	2.19	2.17	2.14
Paper2	82,199	2.61	2.45	2.14	2.13	2.12	2.09
paper3	46,526	2.91	2.60	2.27	2.15	2.12	2.08
paper4	13,286	3.32	2.79	2.52	2.19	2.17	2.14
paper5	11,954	3.41	3.00	2.80	2.48	2.47	2.45
paper6	38,105	2.73	2.54	2.38	2.24	2.24	2.21
progc	39,611	2.67	2.54	2.44	2.32	2.33	2.31
progl	71,646	1.88	1.78	1.76	1.70	1.70	1.7
trans	93,695	1.63	1.53	1.46	1.70	1.68	1.7
Average BPC		**2.62**	**2.34**	**2.19**	**2.16**	**2.14**	**2.12**

It has been observed that, in most of the cases, a better compression is achieved by using RIDBE as the preprocessing stage for the BWT based compressor. Taking the average BPC for all the text files the results can be summarised as follows:

- An improvement of 19.08% is recorded when we take into reckoning the comparison between the average BPC using simple BWT which is 2.62 and using BWT with RIDBE which is 2.12.
- The average BPC using BWT with *-encode is 2.34 and with BWT with RIDBE the average BPC is 2.12, and so the overall improvement is 9.40%.
- The average BPC using BWT with IDBE is 2.19, and with BWT with RIDBE the average BPC is 2.12, which is 3.20% improvement.
- An improvement of 1.85% is logged when we take into account the comparison between the average BPC using BWT with EIDBE which is 2.16 and using BWT with RIDBE the average BPC is 2.12.
- The average BPC using BWT with IIDBE is 2.14, and with BWT with RIDBE the average BPC is 2.12, which is about 1% improvement.

The improvement in average BPC results of RIDBE in comparison with Simple BWT, BWT with *-encoding, BWT with IDBE, BWT with EIDBE and BWT with IIDBE is shown in Figure 2.

5. CONCLUSION

It has been our earnest endeavor, in this paper, to propose a reversible lossless text transformation called Reinforced Intelligent Dictionary Based Encoding (RIDBE). The final

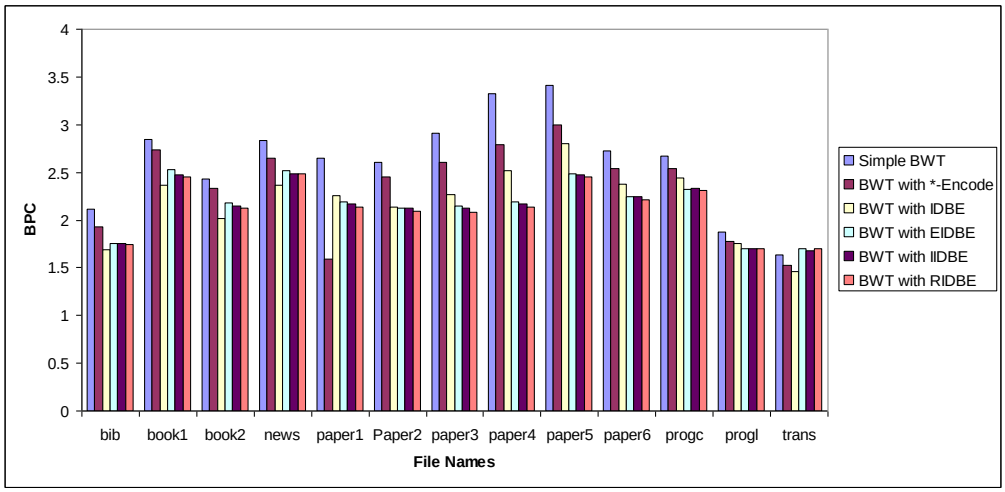

Fig 2. Chart showing the efficient comparison of RIDBE representing BPC comparison of Simple BWT, BWT with *-Encode, BWT with IDBE, BWT with EIDBE and BWT with IIDBE.

results as shown in Table 1 are a clear indicator to the fact that there is a significant improvement in text data compression. Also, the coding mechanism involved helps maintaining confidentiality between the encoder and the decoder as the dictionary used is known only to them. There will be a reduction in the time of the transmission of data, which is due to the marked improvement in data compression. The employment of more efficient and reinforced dictionary is seen to enhance the efficiency of sorting data before compression through preprocessing stage. RIDBE shows an improvement of 19.08% over Simple BWT, 9.40% improvement over BWT with *-encode, 3.20% improvement over BWT with IDBE, 1.85% over BWT with EIDBE and about 1% over BWT with IIDBE. ASCII characters 65 to 90 are the codes used for words in the dictionary aiming towards better compression and to exploit the natural redundancy of the language. There are a few limitations to this area of research, notwithstanding the advantages already enumerated. It is identified that the scope for the successful compression is achieved where majority of the characters are alpha numeric and wherever the non alpha numeric characters are used the compression rate is little less. There is a scope for further research that may lead to improved Data compression.

6. REFERENCES

[1] Arnavut. Z, "Move-to-Front and Inversion Coding", *Proceedings of Data Compression Conference*, IEEE Computer Society, Snowbird, Utah, March 2000, pp. 193- 202.

[2] Awan S, Mukherjee A, "LIPT: A Lossless Text Transform to improve Compression", IEEE, 2001, pp.452-460.

[3] Balkenhol. B, Kurtz. S, and Shtarkov Y.M, "Modifications of the Burrows Wheeler Data Compression Algorithm", *Proceedings of Data Compression Conference*, IEEE Computer Society, Snowbird Utah, March 1999,pp. 188-197.

[4] Balkenhol B and Shtarkov Y, "One attempt of a compression algorithm using the BWT", *SFB343: Discrete Structures in Mathematics*, Faculty of Mathematics, University of Bielefeld, Germany, 1999.

[5] Burrows M and Wheeler D.J, "A Block – sorting Lossless Data compression Algorithm", SRC Research report 124, Digital Research Systems Research Centre, 1994.

[6] Chapin B., and Tate S, "Preprocessing Text to Improve Compression Ratios", *Proceedings of the IEEE Data Compression Conference 1998*, Snowbird, p. 532.

[7] Chapin B, "Higher Compression from the Burrows–Wheeler Transform with new Algorithms for the List Update Problem", Ph.D. dissertation, Department of Computer Science, University of North Texas, 2001.

[8] Cleary J G., Teahan W J., and Ian H. Witten, "Unbounded Length Contexts for PPM', *Proceedings of Data Compression Conference*, IEEE Computer Society, Snowbird Utah, March 1995, pp. 52-61.

[9] Effros M, "PPM Performance with BWT Complexity: A New Method for Lossless Data Compression", Proceedings of Data Compression Conference, IEEE Computer Society, Snowbird Utah, March 2000, pp. 203-212.

[10] Franceschini R, Mukherjee A, "Data Compression using Encrypted Text", IEEE Proceedings of ADL, 1996, pp.130-138.

[11] Horspool N and Cormack G, "Constructing Word–Based Text Compression Algorithms", *Proceedings of the IEEE Data Compression Conference 1992*, Snowbird, pp. 62–71.

[12] Kruse H and Mukherjee A, "Improving Text Compression Ratios with the Burrows–Wheeler Transform", *Proceedings of the IEEE Data Compression Conference 1999*, Snowbird, p. 536.

[13] Kruse H, Mukherjee A, "Preprocessing Text to improve Compression Ratios", Proc. Data Compression Conference, 1998, IEEE Computer Society Press, 1997, p.556.

[14] Moffat A, "Implementing the PPM Data compression scheme", IEEE Transaction on Communications, Vol. 38, No. 11, 1917-1921, 1990.

[15] Radu Radescu, "Transform methods used in Lossless compression of text files", Romanian Journal of Information Science and Technology", Volume 12, Number 1, 2009, 101 – 115.

[16] Robert L and Nadarajan R, "Simple lossless preprocessing algorithms for better compression", The Institution of Engineering and Technology, IET Software 2009, vol 3, pp.37-45

[17] Sadakane K, Okazaki T, and Imai H, "Implementing the Context Tree Weighting Method for Text Compression", *Proceedings of Data Compression Conference*, IEEE Computer Society, Snowbird Utah, March 2000, pp. 123-132

[18] Senthil S, Robert L, "Text Preprocessing using Enhanced Intelligent Dictionary Based Encoding (EIDBE)", Proceedings of Third International Conference on Electronics Computer Technology, April 2011,pp.451-455.

[19] Senthil S, Robert L, "IIDBE: A lossless text transform for better compression", International Journal of Wisdom based computing, August 2011, Volume1(2),pp.1-6.

[20] Seward J, "On the Performance of BWT Sorting Algorithms", *Proceedings of Data Compression Conference*, IEEE Computer Society, Snowbird Utah, March 2000, pp. 173-182.

[21] Shajeemohan B.S, Govindan V.K, 'Compression scheme for faster and secure data transmission over networks', IEEE Proceedings of the International conference on Mobile business, 2005.

[22] Sun W., Zhang N., Mukherjee A. "Dictionary-based fast transform for better compression", proc. IEEE Int. Conf. Information Technology: Coding and Computing, Las vegas, 2003.

[23] Witten I H., Moffat A, Bell T, "Managing Gigabyte, Compressing and Indexing Documents and Images", 2nd Edition, Morgan Kaufmann Publishers, 1999.

[24] Ziv J and Lempel A, "A Universal Algorithm for Sequential Data Compression," IEEE Transactions on Information Theory, pp. 3, 1977.