P. Jeyanthi, V. Anuratha / International Journal of Engineering Research and Applications (IJERA) ISSN: 2248-9622 www.ijera.com
Vol. 2, Issue 5, September- October 2012, pp.807-814

Bwt Based Lossless Reversible Transformation Algorithms – An Analysis

P. Jeyanthi*, V. Anuratha**
*(Research Scholar in Computer Science, Sree Saraswathi Thyagaraja College, Pollachi -642 107, Tamilnadu)
** (Asst. Prof, PG Department of Computer Applications, Sree Saraswathi Thyagaraja College, Pollachi -642 107, Tamilnadu, India)

ABSTRACT

This paper presents and analyzes the benefits provided in lossless compression by using various preprocessing methods that takes advantage of redundancy of the source file. Textual data holds a number of properties that can be taken into account in order to improve compression. Pre-processing deals with these properties by applying a number of transformations that make the redundancy "more visible" to the compressor. Here our focus is on the Burrows-Wheeler Transformation (BWT), Star Transformation, Intelligent Dictionary Based Encoding (IDBE), Enhanced Intelligent Dictionary Based Encoding (EIDBE) and Improved Intelligent Dictionary Based Encoding (IIDBE). The algorithms are briefly explained before calling attention to their analysis.

Keywords - BWT, EIDBE, IDBE and IIDBE

I. INTRODUCTION

While technology keeps developing, the world keeps minimizing. It would be an understatement to merely term this transformation as a technological growth; rather it should be termed as a technological explosion. It is in fact charming to figure out that data compression and its wide techniques have smoothed the progress of this transformation. The amplified spread of computing has led to a massive outbreak in the volume of data to be stored on hard disk and transmitted over the internet. And so it is inevitable that the massive world of internet has to extensively employ data compression techniques in innumerable ways.

Data compression is one of the very exciting areas of computer science. Almost every type of computer users, from students to the business-sector industries depend on data compression techniques to store as much data as possible and maximize the use of storage devices. Data compression is the process of encoding the data in such a way that, fewer bits are needed to represent the data than the original data and thus reducing its size. This process is carried out by means of specific encoding schemes. The key objective is to reduce the physical capacity of data.

The text compression techniques have grabbed the attention more in the recent past as there has been a massive development in the usage of internet, digital storage information system, transmission of text files, and embedded system usage.

Though there are copious methods existing, however, none of these methods has been able to reach the theoretical best-case compression ratio consistently, which suggests that better algorithms may be possible. One approach to attain better compression ratios is to develop different compression algorithms. A number of sophisticated algorithms have been proposed for lossless text compression of which Burrows Wheeler Transform (BWT) [1] and Prediction by Partial Matching [2] outperform the classical algorithms like Huffman, Arithmetic and LZ families [3] of Gzip and Unix – compress [4]. PPM achieves better compression than almost all existing compression algorithms but the main problem is that it is intolerably slow and also consumes large amount of memory to store context information. BWT sorts lexicographically the cyclic rotations of a block of data generating a list of every character and its arbitrarily long forward context. It utilizes Move-To-Front (MTF) [5] and an entropy coder as the backend compressor. Efforts have been made to improve the efficiency of PPM [6], [7], [8] and BWT [5], [9], [10].

An alternative approach, however, is to develop generic, reversible transformations that can be applied to a source text that improves an existing compression algorithm's ability to compress. Thus Preprocessing techniques shows the face in to the picture.

Several significant observations could be made regarding this model. The transformation has to be perfectly reversible, in order to keep the lossless feature of text compression [2]. The compression and decompression algorithms remain unchanged, thus they do not exploit the transformation-related information during the compression [8], [3]. The goal is to boost the compression ratio compared to the results obtained by using the compression algorithm only. Thus these techniques achieve much better compression ratio. These notions are clearly depicted in the figure.

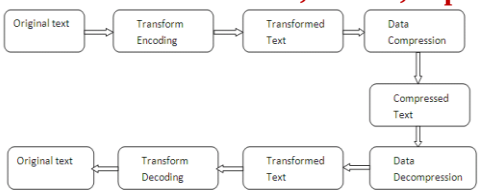

Figure 1. Text compression paradigm incorporating a lossless, reversible transformation

Text preprocessing algorithms are reversible transformations, which are performed before the actual compression scheme during encoding and afterwards during decoding. The original text is offered to the transformation input and its output is the transformed text, further applied to an existing compression algorithm. Decompression uses the same methods in the reverse order: decompression of the transformed text first and the inverse transform after that. Since textual data make up a substantial part of the internet and other information systems, efficient compression of textual data is of significant practical interest.

In the following sections we explain the Burrows-Wheeler Transform (BWT), Star Transform, Intelligent Dictionary Based Encoding (IDBE), Enhanced Intelligent Dictionary Based Encoding (EIDBE) and finally Improved Intelligent Dictionary Based Encoding (IIDBE), followed by their Experimental Results and the last section contains the conclusion remarks.

II. BURROWS-WHEELER TRANSFORMATION

BWT was introduced in 1994 by Michael Burrows and David Wheeler. The following information and analysis data are derived from their work, for this paper. As declared by them, the BWT is an algorithm that procures a block of data and restructures it using a sorting algorithm, then piped through a Move-T0-Front(MTF) stage, then the Run Length encoder Stage and finally an entropy encoder (Huffman coding or Arithmetic Coding) [11]. This is shown in Fig 2. The output block that results from BWT contains exactly the same data element that is fed as input but with differing only in their ordering. This transformation is reversible, which means that the actual ordering of the data elements can be reestablished without losing its fidelity.

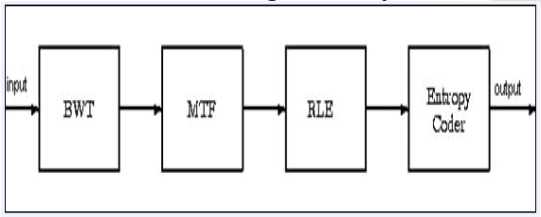

Figure 2. Algorithms following BWT in sequence

A good number of well-known lossless compression algorithms in our day, functions in streaming mode, reading a single byte or a few bytes at a time. But the BWT is applied on an entire block of data at once, and hence also called Block Sorting algorithm. This algorithm is based on a permutation of the input sequence of data which groups symbols with a similar context close together.

If the original string had quite a few substrings that occurred often, then the transformed string will have several places where a single character is repeated multiple times in a row [9]. By applying techniques such as move-to-front transform and run-length encoding, as it is easy to compress a string that has runs of repeated characters, the above mentioned transformation is incredibly useful for compression. The transform is accomplished by sorting all rotations of the text in lexicographic order, then taking the last column. In order to perform the BWT, the first thing we do is treat a string *S*, of length *N*, as if it actually contains N different strings, with each character in the original string being the start of a specific string that is N bytes long. We also treat the buffer as if the last character wraps around back to the first.

In the following example, the text "^BANANA|" is transformed into the output "BNN^AA|A" through these steps (the red | character indicates the 'EOF' pointer):

Transformation				
Input	All Rotations	Sorting All Rows in Alphabetical Order by their first letters	Taking Last Column	Output Last Column
^BANANA|	^BANANA| |^BANANA A|^BANAN NA|^BANA ANA|^BAN NANA|^BA ANANA|^B BANANA|^	ANANA|^B ANA|^BAN A|^BANAN BANANA|^ NANA|^BA NA|^BANA ^BANANA| |^BANANA	ANANA|^B ANA|^BAN A|^BANAN BANANA|^ NANA|^BA NA|^BANA ^BANANA| |^BANANA	BNN^AA|A

Figure 3. Illustration for BWT

The notable aspect about the BWT is primarily not that it generates a more easily encoded output, but that it is reversible, allowing the original document to be restored from the last column data.

III. STAR TRANSFORMATION

The Star Encoding again, is one of the ways to achieve lossless, reversible transformation [12]. This transformation does not compress the text, but prepares it for compression. Star encoding works by creating a large dictionary of frequently used words supposed to be present in the input files. The dictionary must be prepared beforehand and must be known to both the compressor and decompressor. Each word in the dictionary has a star-encoded equivalent, in which as many letters as possible are replaced by the "*" character. The paradigm showing Star Encoding is as follows:

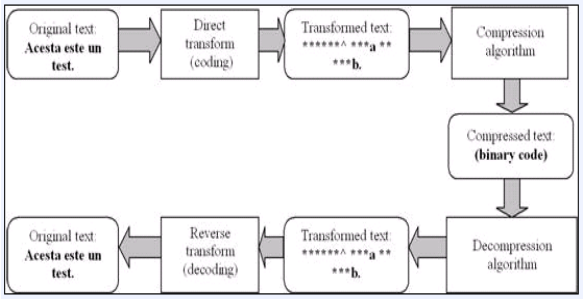

Figure 4. Paradigm incorporating Star Encoding

For example, a commonly used word such as "the" might be replaced by the string "t**". The star-encoding transform simply replaces every occurrence of the word "the" in the input file with "t**". Thus the code string for a word is the same length as the word.

Consequently, the most common words have the highest percentage of "*" characters in their encodings. And thus if the process carried out appropriately, the transformed file will have a vast number of "*" characters. Accordingly, with many runs of stars, the encoded text may be much more compressible than the original text.

Thus the existing star encoding does not provide any compression as such but provide the input text a better compressible format for a later stage compressor. The star encoding is very much weak and vulnerable to attacks.

As an example, a section of text from Project Guttenburg's version of *Romeo and Juliet* looks like this in the original text:

But soft, what light through yonder window breaks?
It is the East, and Iuliet is the Sunne,
Arise faire Sun and kill the enuious Moone,
Who is already sicke and pale with griefe,
That thou her Maid art far more faire then she

Running this text through the star-encoder yields the following text:

B** *of*, **a* **g** *****g* ***d*r ***do* b*e***?
It *s *** E**t, **d ***i** *s *** *u**e,
A***e **i** *un **d k*** *** e****** M****,
*ho *s a****** **c*e **d **le ***h ****fe,
***t ***u *e* *ai* *r* f*r **r* **i** ***n s**

We can clearly see that the encoded data has exactly the same number of characters, but is dominated by stars [14].

IV. INTELLIGENT DICTIONARY BASED ENCODING (IDBE)

Intelligent Dictionary Based Encoding, an encoding strategy offers higher compression ratios and rate of compression. It is observed that a better compression is achieved by using IDBE as the preprocessing stage for the BWT based compressor. There is an immense lessening in the transmission time of files [13], [14].

IDBE comprises of two stages,

Step1: Make an intelligent dictionary

Step2: Encode the input text data

For creation of the dictionary, words are extracted from the input files and ASCII characters 33-250 are assigned as the code for the first 218 words. Likewise for the remaining words the code is assigned as the permutation of two ASCII characters in the range of 33-250. If needed, this assignment moves on to permutation of three and four too. In the course of encoding, the length of the token is determined and it precedes the code. The length is represented by ASCII characters 251-254 with 251 for a code of length 1; 252 for length 2 and so on. The algorithm for encoding and dictionary making [14] is summed up here.

Dictionary Making Algorithm

Start constructing dictionary with multiple source files as input

1. Extract all words from input files.
2. If a word is already in the table
   increment the number of occurrence by 1,
   else
   add it to the table and set the number occurrence to 1.
3. Sort the table in descending order of their frequency of occurrences.
4. Start assigning codes to words in the following method:
   i). Give the first 218 words the ASCII characters 33 to 250 as the code.
   ii). Now give the remaining words each one permutation of two of the ASCII characters (in the range 33-250), taken in order. If there are any remaining words give them each one permutation of three of the ASCII characters and finally if required permutation of four characters.
5. Create a new table having only words and their codes. Store this table as the Dictionary in a file.
6. Stop.

Encoding Algorithm

Start encoding input file

A. Read the dictionary and store all words and their codes in a table

B . While input file is not empty

1. Read the characters from it and form tokens.
2. If the token is longer than 1 character, then
1. Search for the token in the table
2. If it is not found,
1. Write the token as such in to the output file.
   Else
1. Find the length of the code for the word.
2. The actual code consists of the length concatenated with the code in the table, the length serves as a

marker while decoding and is represented by the ASCII characters 251 to 254 with 251 representing a code of length 1; 252 for length 2 and so on.
3. Write the actual code into the output file.
4. Read the next character and neglect that if it is a space. If it is any other character, make it the first character of the next token and go back to B, after inserting a marker character (ASCII 255) to indicate the absence of a space.
Endif
Else
1. Write the 1 character token
2. If the character is one of the ASCII characters 251-255, write the character once more so as to show that it is part of the text and not a marker
Endif
End (While)
C. Stop.

V. ENHANCED INTELLIGENT DICTIONARY BASED ENCODING (EIDBE)

As in IIDBE, the algorithm Enhanced Intelligent Dictionary Based Encoding is also a two step process, with first making an intelligent dictionary and encoding the input data. In comparison with IDBE, EIDBE has been improvised in many aspects. For instance, in IDBE only the first 218 words are assigned single ASCII character representation and the marker character. Whereas in EIDBE, words in the input text are categorized as two letter, three letter and so on up to twenty two letter words. And the first 198 words in each segment have single ASCII character representation and a marker character. The calculation reveals that from a two letter word to a twenty two letter word, single ASCII character representation could be achieved for 4158 words, which is phenomenal compared to IDBE [15].

The dictionary is constructed by extracting words from the input files. The words are then sorted by length in ascending order, followed by sorting on frequency of occurrence in descending order. For the first 198, two letter words, the ASCII characters 33 – 231 are assigned s the code. Code assigning for the rest of the tokens is same as in IDBE. 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 [16]. These details are demonstrated in the following algorithms:

Dictionary Creation Algorithm

Start Creating Dictionary with source files as input
1. Extract words from the input files and check whether it is already available in the table. If it is already available, increment the number of occurrences by one; otherwise add it to the table and set the number of occurrence to one.
2. Sort the table in ascending order of the length of the words.
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 52 (Two letter) words the ASCII characters 65 – 90 and 97 –122 as the code.
   - Now assign each of the remaining words permutation of two of the ASCII characters in the range of 65 – 90 and 97 – 122 taken in order.
   - If any words remain without assigning ASCII characters assign each of them permutation of three of the ASCII characters and finally, if required, permutation of four of the ASCII 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 [16].
6. The created file which consists of only words and their codes serves as the dictionary file.

STOP

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 a space, a maker 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

VI. IMPROVED INTELLIGENT DICTIONARY BASED ENCODING (IIDBE)

Improved Intelligent Dictionary Based Encoding is again a better 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. Decoding is practically feasible too.
In this encoding method, two operations come in to being for the first stage of preprocessing, as transforming the text into some intermediate form with Improved Intelligent Dictionary Based Encoding (IIDBE) scheme and encoding of the transformed text 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 algorithm that has been developed is a two stepped process, the first is making an intelligent dictionary and the next is encoding the input data. The dictionary is constructed by extracting words from the input files. The words are then sorted by length in ascending order, followed by sorting on frequency of occurrence in descending order. For the first 52, two letter words, the ASCII characters 65 – 90 and 97 – 122 are assigned as code. The remaining words if any are coded as permutation of two ASCII characters in the same range as mentioned before, followed by three ASCII characters if needed and finally four ASCII characters if still more words remain. The same course of action is repeated for three letter words, four letter words and so on up to twenty two letter words, as the maximum length of words in English is 22. In the actual code, the length of the word is 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 [17]. Thus there is no major change in the algorithm of IIDBE than EIDBE except the range of ASCII characters. But it shows a remarkable improvement in the compression rate than that of EIDBE as shown in TABLE III.

VII. EXPERIMENTAL RESULTS

For purpose of comparison, the following table shows the raw size of some of the files from Calgary corpus, the compressed sizes using the BWT compressor and the compressed sizes using PKZIP.

Table I.
Mandatory Comparison of BWT with PKZIP

File Name	Raw Size	PKZIP Size	PKZIP Bits/Byte	BWT Size	BWT Bits/Byte
bib	111,261	35,821	2.58	29,567	2.13
book1	768,771	315,999	3.29	275,831	2.87
book2	610,856	209,061	2.74	186,592	2.44
geo	102,400	68,917	5.38	62,120	4.85
news	377,109	146,010	3.10	134,174	2.85
obj1	21,504	10,311	3.84	10,857	4.04
obj2	246,814	81,846	2.65	81,948	2.66
progp	49,379	11,248	1.82	11,404	1.85
trans	93,695	19,691	1.68	19,301	1.65
Total	2,381,789	898,904	3.0	811,794	2.8

From the table with a list of files, it is clear that BWT achieves compression pretty well when compared with the commercial product PKZIP, as the average bits needed to represent a single byte of data in BWT is 2.8, whereas in PKZIP, it is 3.0

In the following table the performance issues such as Bits Per Character (BPC) and conversion time are compared for the three cases i.e., simple BWT, BWT with Star encoding and BWT with Intelligent Dictionary Based Encoding (IDBE).

Table II.
BPC comparison of simple BWT, BWT with *Encode and BWT with IDBE in Calgary corpuses

Calgary corpuses							
File Names	File size Kb	BWT		BWT with *Encode		BWT with IDBE	
		BPC	Time (Secs)	BPC	Time (Secs)	BPC	Time (Secs)
bib	108.7	2.11	1	1.93	6	1.69	4
book1	750.8	2.85	11	2.74	18	2.36	11
book2	596.5	2.43	9	2.33	14	2.02	10
geo	100.0	4.84	2	4.84	6	5.18	5
news	368.3	2.83	6	2.65	10	2.37	7
paper1	51.9	2.65	1	1.59	5	2.26	3
paper2	80.3	2.61	2	2.45	5	2.14	4
paper3	45.4	2.91	2	2.60	6	2.27	3
Paper4	13.0	3.32	2	2.79	5	2.52	3
Paper5	11.7	3.41	1	3.00	4	2.8	2
Paper6	37.2	2.73	1	2.54	5	2.38	3
progc	38.7	2.67	2	2.54	5	2.44	3
prog1	70.0	1.88	1	1.78	5	1.70	3
trans	91.5	1.63	2	1.53	5	1.46	4

The results are shown graphically too and prove that BWT with IDBE out performs all other techniques in compression ratio and speed of compression (conversion time).

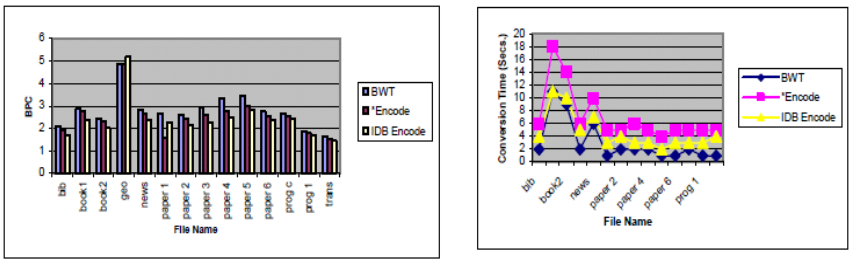

Figure 4. BPC & Conversion time comparison of transform with BWT, BWT with *Encoding and BWT with IDBE for Calgary corpus files.

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

Table III
BPC COMPARISON OF IIDBE AND EIDBE WITH SIMPLE BWT, BWT WITH STAR ENCODING, BWT WITH IDBE IN CALGARY CORPUS

File Names	File size in bytes	Simple BWT	BWT With* Encode	BWT with IDBE	BWT with EIDBE	BWT with IIDBE
Bib	1,11,261	2.11	1.93	1.69	1.76	1.76
book1	7,68,771	2.85	2.74	2.36	2.53	2.47
book2	6,10,856	2.43	2.33	2.02	2.18	2.15

news	3,77,109	2.83	2.65	2.37	2.52	2.49
paper1	53,161	2.65	1.59	2.26	2.19	2.17
Paper2	82,199	2.61	2.45	2.14	2.13	2.12
paper3	46,526	2.91	2.60	2.27	2.15	2.12
paper4	13,286	3.32	2.79	2.52	2.19	2.17
paper5	11,954	3.41	3.00	2.80	2.48	2.47
paper6	38,105	2.73	2.54	2.38	2.24	2.24
progc	39,611	2.67	2.54	2.44	2.32	2.33
progl	71,646	1.88	1.78	1.76	1.70	1.70
trans	93,695	1.63	1.53	1.46	1.70	1.68
Average BPC		**2.62**	**2.34**	**2.19**	**2.16**	**2.14**

It has been observed that, in most of the cases, a better compression is achieved by using IIDBE as the preprocessing stage for the BWT based compressor.

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

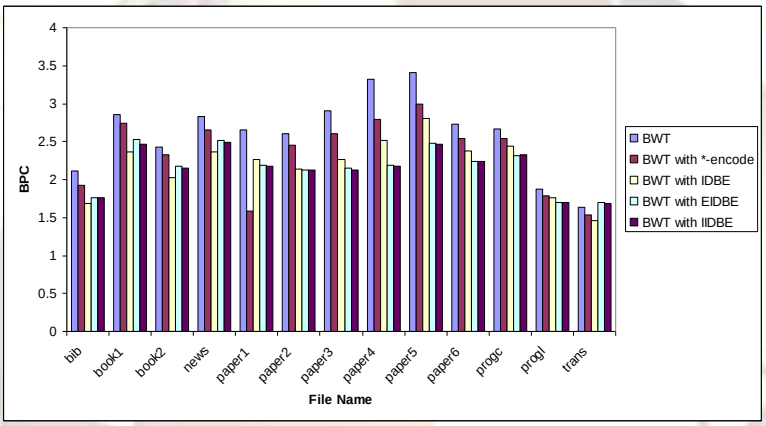

Figure 6. Chart showing the efficient comparison of IIDBE representing BPC comparison of Simple BWT, BWT with *-Encode, BWT with IDBE and BWT with IIDBE.

VIII. CONCLUSION

In this paper we have analyzed the reversible lossless text transformation algorithms BWT, Star Transformation, Intelligent Dictionary Based Encoding (IDBE), Improved Intelligent Dictionary Based Encoding (IIDBE) and finally Enhanced Intelligent Dictionary Based Encoding (EIDBE). We also submitted the performance evaluation of these transformations on the standard set of files from Calgary corpus as achieved by various authors. The final results as shown in Table III points to a significant improvement in text data compression. IIDBE shows an improvement of 18.32% over Simple BWT, 8.55% improvement over BWT with *-encode, 2.28% improvement over BWT with IDBE and about 1% over BWT with EIDBE.

REFERENCES

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

[2] Moffat A, "Implementing the PPM Data compression scheme", IEEE Transaction on Communications, 38(11): 1917-1921, 1990.

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

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

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

[6] 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

[7] 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.

[8] 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

[9] 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.

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

[11] Rexlin. S. J , Robert. L , "Dictionary Based Preprocessing Methods in Text Compression – A Survey", International Journal of Wisdom Based Computing, Vol. 1 (2), August 2011

[12] Franceschini R., Kruse H., Zhang N., Iqbal R., Mukherjee A., Lossless, Reversible Transformations that Improve Text Compression Ratios, Preprint of the M5 Lab, University of Central Florida, 2000.

[13] Govindan. V. K, Shajee Mohan. B. S, "IDBE - An Intelligent Dictionary Based Encoding Algorithm for Text", 2006

[14] 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.

[15] 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.

[16] 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.

[17] Senthil. S, Robert. L, "IIDBE: A Lossless Text Transform for Better Compression", 2012