Volume 3, No. 8, August 2012
Journal of Global Research in Computer Science
ISSN-2229-371X
REVIEW ARTICAL
Available Online at www.jgrcs.info

ANALYSIS OF LOSSLESS REVERSIBLE TRANSFORMATION ALGORITHMS TO ENHANCE DATA COMPRESSION

P. Jeyanthi[1*], V. Anuratha[2]
[*1]Research Scholar in Computer Science, Sree Saraswathi Thyagaraja College, Pollachi -642 107, Tamilnadu, India
jeyanthi_mahen@yahoo.com[1]
[2]Asst. Prof, PG Department of Computer Applications, Sree Saraswathi Thyagaraja College, Pollachi -642 107, Tamilnadu, India
mailanuvinu@yahoo.co.in[2]

Abstract **–** In this paper we analyze and present the benefits offered in the lossless compression by applying a choice of preprocessing methods that exploits the 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 cope up with these properties by applying a number of transformations that make the redundancy "more visible" to the compressor. Many pre-processing algorithms come into being for text files which complement each other and are performed prior to actual compression. Here our focus is on the Length-Index Preserving Transform (LIPT), its derivatives ILPT, NIT & LIT and StarNT Transformation algorithm. The algorithms are briefly presented before calling attention to their analysis.

Keywords **-** LIPT, ILPT, NIT, LIT and StarNT

INTRODUCTION

The amplified spread of computing has led to a massive outbreak in the volume of data to be stored on hard disks and sent over the Internet. This escalation has led to a crucial need for "Data compression" which is the process of encoding information using fewer bits than the original representation would use. It is the ability of reducing the amount of storage or Internet bandwidth required to handle this data. The most vital objective of any compression algorithm is the compression efficiency. Intuitively, the behavior of a compression algorithm would depend on the data and their internal structure. The more redundancy the source data has, the more effective a compression algorithm may be.

The vital feature of merit for data compression is the "compression ratio", which is the ratio of the size of a compressed file to the original uncompressed file. For example, suppose a data file takes up 30 kilobytes (KB). Using data compression techniques, the file could be reduced in size to, say, 15 KB that makes it easier to store on disk and helps faster transmission over an Internet connection. Thus the data compression software reduces the size of the data file in this case by a factor of two, and hence the "compression ratio" of 2:1 is attained. Thus 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 the size of the data. This process is carried out by means of specific encoding schemes.

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

Though there are bountiful 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) [4] and Prediction by Partial Matching [12] outperform the classical algorithms like Huffman, Arithmetic and LZ families [22] of Gzip and Unix – compress [20]. 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) [1] and an entropy coder as the backend compressor. Efforts have been made to improve the efficiency of PPM [6], [8], [17] and BWT [1], [3], [18].

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

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 [12]. The compression and decompression algorithms remain unchanged, thus they do not exploit the transformation-related information during the compression [17], [22]. These notions are clearly depicted in the Fig1. The goal is to boost the compression ratio in comparison with that obtained using the compression algorithms alone. Thus these techniques achieve much better compression ratio.

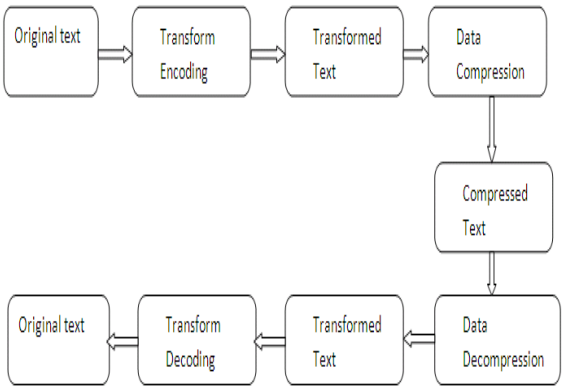

Figure. 1 Text compression paradigm incorporating a lossless, reversible transformation

As shown in the above fig, 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 subsequent sections we put in words the Length Index Preserving Transformation (LIPT) [9], Initial Letter Preserving Transform (ILPT), Numerical Index Transform (NIT), Literal Index Transform (LIT) [16] and finally the StarNT [19] Transform. The last section holds the conclusion remarks.

LENGTH INDEX PRESERVING TRANSFORM (LIPT)

The core concept of compression is to transform the text into some intermediate form which can be compressed with better efficiency and which exploits the natural redundancy of the language in making this transformation. LIPT encoding scheme by Fauzia S. Awan and Amar Mukherjee [9], [2] makes use of recurrence of same length of words in the English language to create context in the transformed text that the entropy coders can exploit. LIPT uses letters of the alphabet to denote lengths of the words. Hence these letters will be repeated again and again in the transformed text resulting in better context. In addition to this, LIPT also uses the letters of the alphabet to denote the offset within a block of words in the English dictionary having the same length. This serves to induce additional context in the transformed text.

Word frequency data from Calgary [5], Canterbury [5] and Gutenberg Corpus [10] are collected to support the central theme of repetition of length of words in English text. From the Fig.2 it is clear that the maximum number of words has length 3 and most words lie in the range of length 2 to 9 after which the word frequency comes to negligible level.

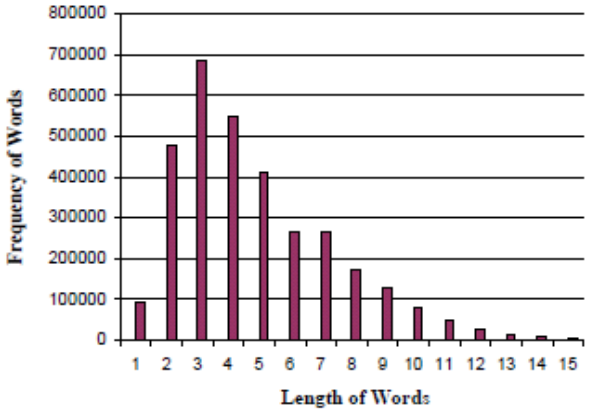

Figure: 2 Frequency of words Vs length of words in test corpus

LIPT comprises of two steps,
Step1: Make an efficient dictionary
Step2: Encode the input text data

The description of LIPT is as follows. A dictionary *D* of words in the corpus is partitioned into disjoint dictionaries *Di*, each containing words of length *i*, where $i = 1,2\ldots n$. Each dictionary *Di* is partially sorted according to the frequency of words in the corpus. Then a mapping is used to generate the encoding for all words in each dictionary *Di*. *Di*[*j*] denotes the *j*th word in dictionary *Di*. In LIPT, the *jth* word *Di*[*j*], in the dictionary *D* is represented as $*C_{len}[c][c][c]$ (the square brackets denote the optional occurrence of a character or letter of the alphabet enclosed and are not part of the transformed representation) where C_{len} stands for a character in the alphabet [a-z, A-Z] each denoting a corresponding length [1-26, 27-52] and each *c* cycles through [a-z, A-Z]. If $j = 0$ then the encoding is $* C_{len}$. For $j>0$, the encoding is $* C_{len}\, c[c][c]$. Thus, for $1 £ j £ 52$ the encoding is $* C_{len}c$; for $53 £ j £ 2756$ it is $* C_{len}cc$, and for $2757 £ j £ 140608$ it is $* C_{len}ccc$. Let us denote the dictionary of words containing the transformed words as *DLIPT*. Thus, the 0^{th} word of length 10 in the dictionary *D* will be encoded as "*j" in *DLIPT*, *D*10[1] as "*ja", *D*10[27] as "*jA", *D*10[53] as "*jaa", *D*10[79] as "*jaA", *D*10[105] as "*jba" , *D*10[2757] as "*jaaa", *D*10[2809] as "*jaba", and so on.

The transform must also be able to handle special characters, punctuation marks and capitalization. The character '*' is used to denote the beginning of an encoded word. The character '~' at the end of an encoded word denotes that the first letter of the input text word is capitalized. The character '`' denotes that all the alphabets in the input word are capitalized. A capitalization mask, preceded by the character '^', is placed at the end of encoded word to denote capitalization of alphabets other than the first letter and all capital letters. The character '\' is used as escape character for encoding the occurrences of '*', '~', '`', '^', and '\' in the input text. The decoding process is the reverse of the above mentioned encoding process.

Dictionary Making Algorithm:

a. Start constructing dictionary 'D' (English language dictionary with about 60,000 words taking 0.5 Mb is used here)

b. Partition dictionary 'D' in to disjoint dictionaries D_1, D_2, …, D_n, where D_1 possessing words of length 1, D_2 of length 2 and so on
c. Sort each disjoint dictionary according to their frequency of occurrence
d. Start assigning addresses to the words as
   $C_{len}[c][c][c]$
   Where C_{len} denotes a character in the set [a-z, A-Z], each character representing corresponding length in set [1-26, 27-52]
e. Each *c* cycles through [a-z, A-Z].
f. Handle special characters as mentioned

Encoding steps:

a. The words in the input file are extracted
b. These words are searched in the Dictionary *D* using a two level index search method.
c. If found, its position and block number (*i* and *j* of *Di*[*j*]*)* are noted and the corresponding transformation at the same position and length block in *DLIPT* is looked up. This is the encoding for the respective input word.
d. If the input word is not found in dictionary *D* then it is transferred for output as it is.
e. Once the whole file or the entire text is transformed as in steps 1 and 2, the transformed text is then fed to a compressor (e.g. Bzip2, PPM etc.).

Decoding steps:

a. Using the compressor as was used at the sending end, the received encoded text is first decoded and the transformed LIPT text is recovered.
b. Reverse transformation is then applied on the decompressed transformed text. The words with '*' represent transformed words and those without '*' represent non-transformed words and do not need any reverse transformation. The length character in the transformed words gives the length block and the next three characters give the offset in the respective block and then there might be a capitalization mask. The words are looked up in the original dictionary *D* in the respective length block and at the respective position in that block as given by the offset characters. The transformed words are replaced with the respective English dictionary *D* words.
c. The capitalization mask is applied.

This scheme allows for a total of 140608 encodings for each word length. Since the English words re limited to a maximum length of around 22 and the maximum number of words in any *Di* in the English dictionary is less than 10,000, this scheme deals with all English words in the dictionary.

ILPT, NIT, AND LIT TRANSFORMS

The three transform methods to be portrayed here are all derived by Radu RADESCU [14], [15] from Length-Index Preserving Transform (LIPT), which is presented in the previous section. ILPT, NIT, and LIT do lossless reversible text transforms, and are based upon LIPT Transform [2], [14], [15]. These methods do not offer a significant increase in the execution time performance, because they use the same method of loading a dictionary as LIPT does, and the static dictionary and the code dictionary remain the same.

Initial Letter Preserving Transform (ILPT) is similar to LIPT [8] in all the aspects except the characteristic that the dictionary is sorted in blocks according to the initial letters of the words instead of length. Then descending order of frequencies of occurrence, the words in each block of letters are sorted. Thus the character used for LIPT is the length of the coded word but in the case of ILPT, it is the first letter of the coded word, that is instead of * C_{len}[c][c][c], for ILPT is *C_{init}[c][c][c], where C_{init} represents the first letter of the coded word. Besides that single feature, everything else remains as for LIPT.

Numerical Index Transform (NIT) uses variable addresses based on numbers instead of letters of the alphabet. When this method applied on English dictionary D which is sorted first by length of the words and then the frequency of their appearance, offered a performance inferior to LIPT. And hence, the dictionary was sorted globally in descending order of the frequency of appearance of the words. No sorting of blocks in the newly created dictionary. The transformed words are represented by the character "*" followed by the corresponding code of the respective word. This way the first word is coded as "*0", the 1000^{th} word is coded as "*999", and so on. Special characters are treated the same way as was done in LIPT.

Literal Index Transform (LIT) method is very much similar to NIT, except that here, for the specification of the linear address of a word in the dictionary, the alphabets [a–z; A–Z] are used instead of numbers.

In these transformations, the size of the dictionary of transformation is variable depending on the individual transform. From the observations, it is clear that ILPT has the dictionary of transformation with the smallest size. Another fact is that the frequency of the repeated words remains the same in the original text file and the transformed one, only the frequency of the characters changes. This factor, together with reducing the file size, contributes to a better compression by using these transforms. Also, arranging words in descending order of their frequency of use, leads to use shorter codes for words used more often and longer codes for less used words. This again leads to smaller size of the files.

STARNT TRANSFORMATION

Star New Transform (StarNT), a fast transform algorithm was proposed by Weifeng Sun, Nan Zhang and Amar Mukherjee [19]. This method is superior to LIPT [9] not only in compression performance, but also in time complexity. When bzip2 and PMD assisted with StarNT, both achieves a better compression performance. In this transformation, Ternary Search Tree [11] is applied to accelerate the transform encoding. Searching in Ternary search trees are quite straightforward. Furthermore, ternary search trees are quite space efficient. Figure 3 illustrates a ternary search tree for seven words (a, air, all, an, and, as, at) where only 9 nodes instead of 16 nodes are used.

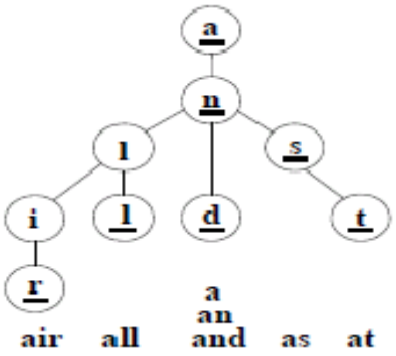

Figure. 3 Illustration of a Ternary Search Tree

In the transform encoding module, words in the transform dictionary are stored in the ternary search tree with the address of corresponding codewords. The ternary search tree is split into 26 distinct ternary search sub-trees. The root addresses of these ternary search sub-trees are stored in an array. Each ternary search tree contains all words with same initial characters. For example, all words with initial character 'a' in the transform dictionary exist in the first ternary search sub-tree, while all words with initial character 'b' exist in the second sub-tree, and so on.

The order in which we insert nodes into the ternary search tree has a lot of performance impact. First, this order determines the time needed to construct the ternary search tree of the transform dictionary. Second, it also determines the performance of the search operation that is the key factor of the transform efficiency. In this transform the natural order of words in the transform dictionary is followed. Results show that this approach works very well.

Dictionary Mapping:

The transform dictionary used is prepared in advance, and shared by both the transform encoding module and the transform decoding module. The words in the transform dictionary ***D*** are sorted according to the following rules:

Most frequently used words are listed in the beginning of the dictionary in the decreasing order of their frequency of occurrence. There are 312 words in this group.

The remaining words are sorted in ***D*** according to their lengths. Words with longer lengths are stored after words with shorter lengths. Words with same length are sorted in the decreasing order of their frequency of occurrence.

To gain a much better compression performance for the backend data compression algorithm, only letters [a..z, A..Z] can be used to represent the codeword.

The first 26 words are assigned "a", "b", …,"z" as their codewords. The next 26 words are assigned "A", "B", …, "Z". The 53rd word is assigned "aa", 54th "ab". Following this order, "ZZ" is assigned to the 2756th word in the Dictionary. The 2757th word is assigned "aaa", the following 2758th word is assigned "aab", and so on. Using this mapping mechanism, totally 52+52*52+52*52*52 = 143,364 words can be included in the Dictionary. Capital conversion technique is also introduced by placing the escape symbol and flag director at the end of the codewords.

Transform Encoding:

In this transformation, the character '*' means that the following word does not exist in the transform dictionary D. The key reason for this change from the earlier Star family is to reduce the size of the transformed intermediate file and thus the encoding/decoding time of the backend compression algorithm can be minimized.

The initial letter capitalized words and all-letter capitalized words are handled by some specialized operations. The character '»' appended to the transformed word denotes that the initial letter of the corresponding word in the original text file is capitalized. The appended character ' ` ' denotes that all letters of the corresponding word in the original text file are capitalized. The character '*n*' is used as escape character for encoding the occurrence of '*', '»', ' ` ' , and '*n*' in the input text file.

Encoding Algorithm:

a. Initiate a Transformer
b. Read the input text
c. If word exist in Transform Dictionary
   Replace word with corresponding codeword
   Append special symbol if necessary
   Else
   Prefix the word with character '*'
d. Continue steps 2 and 3 till end of file

The transform decoding module performs the inverse operation of the transform encoding module.

EXPERIMENTAL RESULTS

In this section the tables showing the comparison between various algorithms are given. In Table I, the compression results in terms of average bits per character (BPC) is given. To support the point of repetition of length of words in English text, the word frequency data from Calgary, Canterbury [5] and Gutenberg corpus [10] are used.

The compression results on text files derived from Canterbury, Calgary [5] and Gutenberg corpus [10] show compression ratio improvement of around 5% for BZip2 with LIPT. The compression results also show an improvement in the range of around 2% to 7% for compression methods with LIPT [14].

Table 1: BPC Comparison between original Bzip2 –9, and Bzip2 –9 with LIPT for the files in three Corpuses

FileNames	Bzip2 (BPC)	Bzip2 with LIPT (BPC)	FileNames	Bzip2 (BPC)	Bzip2 with LIPT (BPC)	FileNames	Bzip2 (BPC)	Bzip2 with LIPT (BPC)
Calgary			**Canterbury**			**Gutenberg**		
paper5	3.24	**2.95**	grammar.lsp	2.76	**2.58**	anne11.txt	2.22	**2.12**
paper4	3.12	**2.74**	xargs.1	3.33	**3.10**	1musk10.txt	2.08	**1.98**
paper6	2.58	**2.40**	fields.c	2.18	**2.14**	world95.txt	1.54	**1.49**
progc	2.53	**2.44**	cp.html	2.48	**2.44**	Average BPC	1.95	**1.86**
paper3	2.72	**2.45**	asyoulik.txt	2.53	**2.42**			
progp	1.74	**1.72**	alice29.txt	2.27	**2.13**			
paper1	2.49	**2.33**	lcet10.txt	2.02	**1.91**			
progl	1.74	**1.66**	plrabn12.txt	2.42	**2.33**			
paper2	2.44	**2.26**	world192.txt	1.58	**1.52**			
trans	1.53	**1.47**	bible.txt	1.67	**1.62**			
bib	1.97	**1.93**	kjv.gutenberg	1.66	**1.62**			
news	2.52	**2.45**	Average BPC	2.26	**2.17**			
book2	2.06	**1.99**						
book1	2.42	**2.31**						
Average BPC	2.36	**2.22**						

From the table, it is clear that LIPT outperforms Bzip2. For example, when we have a look at the average BPC of the selected Calgary files, it is 2.36 for Bzip2 and only 2.22 for Bzip2 with LIPT. This is a remarkable difference

Software compression [7], [13], [15] results are presented for the files using Initial Letter Preservation Transform (ILPT), Literal Index Transform (LIT) and Numerical Index Transform (NIT) with the classic archiver WinRar (see Tables II– IV). For the purpose of evaluation, some representative Romanian text files and two test files, called "book1" and "book2" are taken from the set of evaluation of lossless compression algorithms Calgary Corpus [5].

Table 2: Initial letter preserving transform

File Name	File Size	ILPT Compression	WinRar Compression	
book1.txt	767	596	273	261
book2	612	456	179	161
creierul o enigma	494	328	139	122
regulament ordine interioara	84	50	14	10
tarzan of the apes	497	380	165	144
Yserver	260	158	14	12

Table 3: Numerical Index Transform

File Name	File Size	NIT Compression	WinRar Compression	
book1.txt	767	697	273	233
book2	612	516	179	157
creierul o enigma	494	374	139	118
regulament ordine interioara	84	55	14	10
tarzan of the apes	497	434	165	141
Yserver	260	137	14	10

Table 4: Literal Index Transform

File Name	File Size	LIT Compression	WinRar Compression	
book1.txt	767	571	273	231
book2	612	431	179	154
creierul o enigma	494	310	139	117
regulament ordine interioara	84	55	14	10
tarzan of the apes	497	364	165	140
Yserver	260	124	14	9

In the following graphical representations (Fig 4-9) there is a noticed improvement in the compression of files with increase in the size of the file. As transformation algorithm is based on the exploitation of redundancy, larger the file, better the compression.

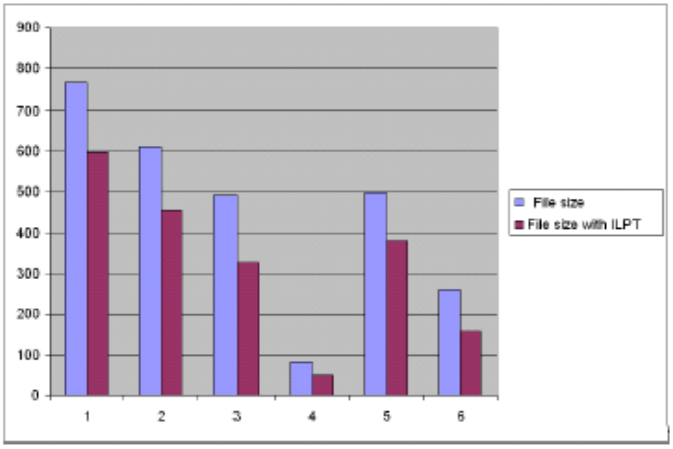

Figure 4. The original and ILPT transformed file

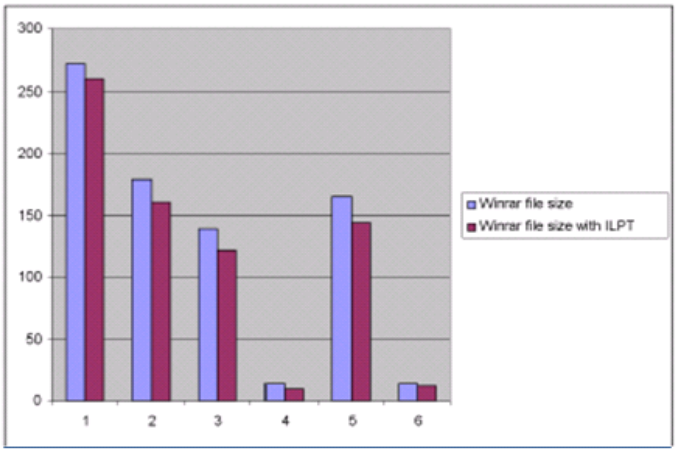

Figure 5. WinRar archived file with and without ILPT

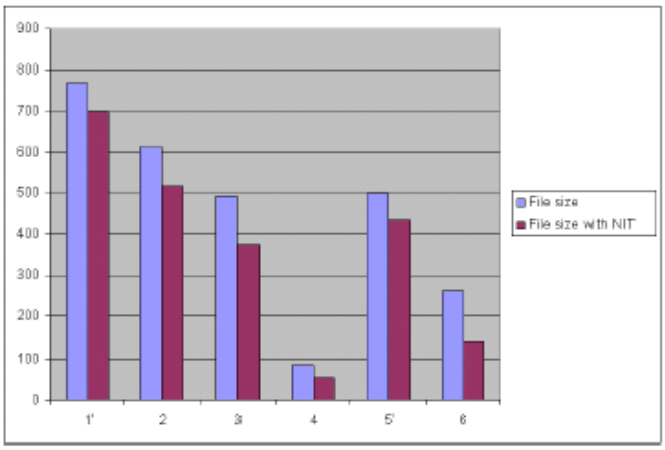

Figure 6. The file with and without NIT

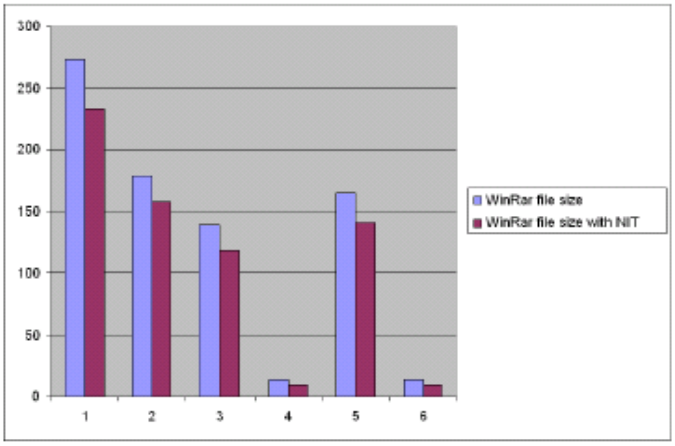

Figure 7. WinRar archived file with and without NIT

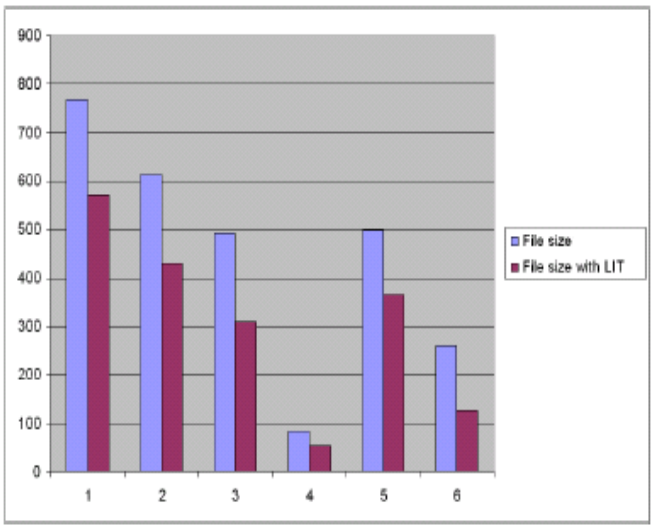

Figure 8. The file with and without LIT

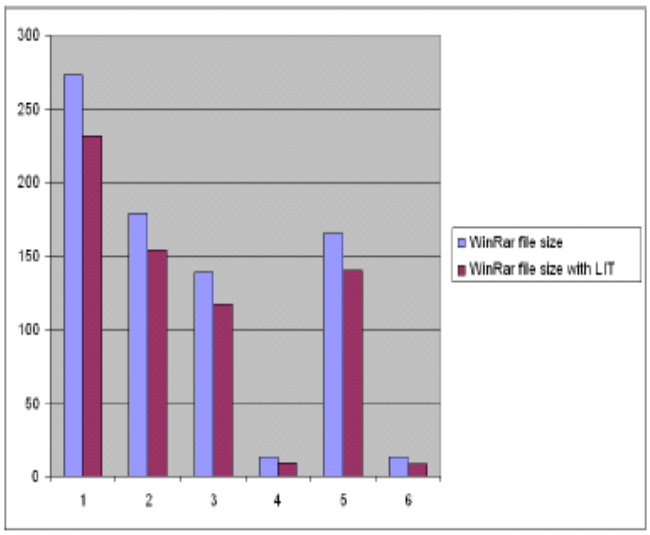

Figure 9. WinRar archived file with and without LIT

Now the experimental data for StarNT is illustrated in Table V. All these data are average values of 10 runs. The results can be summarized as follows:

a. The average transform encoding time using new transform is only about 23.7% of that using LIPT.
b. The average transform decoding time using new transform is only about 15.1% of that using LIPT.
c. Especially, the speed of transform encoding phase and decoding phase of the transform algorithm is asymmetric. The decoding module runs faster than encoding module by 39.3% averagely. The main reason is that the simple address calculating function used in the transform decoding module is more efficient than the ternary search tree used in the transform encoding module.

Table 5: Comparison of Transform Time

Corpora	StarNT		LIPT	
	Transform Encoding	Transform Decoding	Transform Encoding	Transform Decoding
Calgary	0.42	0.18	1.66	1.45
Canterbury	1.26	0.85	5.7	5.56
Gutenburg	1.68	1.12	6.89	6.22
Average	0.89	0.54	3.75	3.58

Table 6: Comparison of Encoding Speed

	Calgry	Cantrbry	Gutnbrg	AVRG
bzip2	0.36	2.73	4.09	1.69
bzip2+StarNT	0.76	3.04	4.40	2.05
bzip2+ LIPT	1.33	5.22	7.01	3.47
gzip	0.23	2.46	2.28	1.33
gzip+StarNT	0.86	3.36	3.78	2.06
gzip+LIPT	1.70	6.59	9.67	4.47
PPMD	9.58	68.3	95.4	41.9
PPMD+StarNT	7.94	55.7	75.2	33.9
PPMD+LIPT	9.98	69.2	90.9	41.9

Table 7: Comparison of Decoding Speed

	Calgry	Cantrbry	Gutnbrg	AVRG
bzip2	0.13	0.82	1.15	0.51
bzip2+StarNT	0.33	1.53	2.22	1.00
bzip2+ LIPT	1.66	6.77	8.46	4.40
gzip	0.04	0.22	0.29	0.14
gzip+StarNT	0.27	1.16	1.44	0.72
gzip+LIPT	1.64	9.15	7.99	5.27
PPMD	9.65	71.2	95.4	43.0
PPMD+StarNT	8.07	57.8	76.9	35.0
PPMD+LIPT	10.9	77.2	98.7	46.4

Now the compression ratio of bzip2+StarNT, bzip2+LIPT along with the results of bzip2 alone is illustrated in Table V. The average compression ratio using only bzip2 algorithm is 2.36 and using the bzip2 algorithm along with the LIPT technique is 2.06 which emphasizes better compression improvement. StarNT compressor shows better compression results when compared with LIPT which gives compression ratio of 1.94%.

Table 8: Comparative Compression Results of Starnt with Lipt

File Names	File size Bytes	Bzip2	Bzip2 +LIPT	Bzip2 +StarNT
bib	111261	1.97	1.93	1.71
book1	768771	2.42	2.31	2.28
book2	610856	2.06	1.99	1.92
News	377109	2.52	2.45	2.29
paper1	53161	2.46	2.33	2.21
paper2	82199	2.44	2.26	2.14
Progc	39611	2.53	2.44	2.32
prog1	71646	1.74	1.66	1.58
Progp	49379	1.74	1.72	1.69
trans	93695	1.53	1.47	1.22
Average BPC		2.36	2.06	1.94

CONCLUSIONS

This paper presents important results in preprocessing for lossless compression algorithms using a set of transforms for different text files. Bzip2 with LIPT shows an improvement of 5.24% over the original Bzip2 –9. Also another important result is that Bzip2 with LIPT is much faster in time performance. When focusing attention over ILPT, NIT and LIT, all three transforms present significant improvements over the original files in terms of

compression rate. There is not a distinction to be seen between transforms, ie., no one can say that one is better than the other. Again it is very remarkable that bzip2 + StarNT could provide a better compression performance that maintains a convincing compression and decompression speed while it is compared with LIPT. StarNT preprocessing skill uses temary search tree to accelerate changes in encoding operation and hashing method in decoding execution to hurry up the transformation. The StarNT works better than LIPT when is applied with backend compressor.

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 F.S., Zhang N., Motgi N., Iqbal R.T., Mukherjee A., LIPT: A Reversible Lossless Text Transform to Improve Compression Performance, Proc. of Data Compression Conf., Snowbird, UT, 2001.

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

[5] Calgary and Canterbury Corpi

http://corpus.canterbury.ac.nz

[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] Derived classes MyTabCtrl: http://www.codersource.net/mfc ctabctrl.html

[8] 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.

[9] Fauzia S. Awan, Amar Mukherjee, "LIPT : A Lossless Text Transform to Improve Compression," In Proceedings of International Conference on Information and Theory : Coding and Computing, Las Vegas, Nevada, 2001. IEEE Computer Society

[10] Gutenberg Corpus

http://www.promo.net/pg/

[11] J. L. Bentley and R. Sedgewick. Fast Algorithms for Sorting and Searching Strings. In Proceedings of the 8th Annual ACM-SIAM Symposium on Discrete Algorithms, New Orleans, January 1997.

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

[13] Ostream libraries: http://www.cplusplus.com/reference/iostream/ostream/

[14] Radescu R., Lossless Text Compression Using the LIPT Transform, Proceedings of the 7th International Conference Communications 2008 (COMM2008), pp. 59–62,Bucharest, Romania, 5–7 June 2008.

[15] Radescu R., Transform Methods Used in Lossless Compression of Text Files, RomanianJournal of Information Science and Technology (ROMJIST), Publishing House of the Romanian Academy, Bucharest, vol. 12, no. 1, pp. 101–115, 2009, ISSN 1453-8245.

[16] Radu RADESCU, "LIPT – Derived Transform Methods used in Lossless compression of Text", U.P.B Sci. Bull., Series C, Vol. 73, Iss. 2, 2011

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

[19] W. Sun, A. Mukherjee, N. Zhang, "A Dictionary-based Multi-Corpora Text Compression System," Proceedings of the 2003 IEEE Data Compression Conference, ,March 2003.

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

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

Short Bio Data for the Author

 **Jeyanthi.P**. Completed B.Sc (Physics) followed by Master of Computer Applications in Madras Christian College, Chennai. Completed Master of Business Administration (IS) in 2010. Worked as Assistant Professor for a period of three years and as IT Admin, Singapore for an year. Currently pursuing Master of Philosophy in the area of Data Compression in Saraswathy Thyagaraja College, Pollachi.

 **Anuratha. V.** Pursued B.Sc (Computer Science) at PSG College of Arts & Science, Coimbatore, during the period of 1992 – 95. Completed Masters Degree in Computer Applications in the year 1999 and Master of Philososphy in 2002. Currently doing Ph.D in the area of Wireless Networks in verge of completion. Published more than 5 Research papers. Guided many Scholars. Currently workig as Assistant Professor in PG Department of Computer Applications, Sree Saraswathy Thyagaraja College, Pollachi