

Annual Report for Period:07/2000 - 06/2001**Submitted on:** 06/08/2001**Principal Investigator:** Mukherjee, Amar .**Award ID:** 9977336**Organization:** U of Central Florida**Title:**
Algorithms to Improve the Efficiency of Data Compression and Caching on Wide-Area Networks

Project Participants

Senior Personnel

Name: Mukherjee, Amar**Worked for more than 160 Hours:** Yes**Contribution to Project:**

Professor Amar Mukherjee is the Principal Investigator of this project and is in charge of all the reserach and targetted activities and guidance of research assitants working under this project.

Post-doc

Graduate Student

Name: Zhang, Nan**Worked for more than 160 Hours:** Yes**Contribution to Project:**

Nan Zhang is working as a Graduate Research Assistant with the project. He is working on developing compression algorithms and a theory of transforms developed under this project. He is also reading literature on compressed domain search problem to come up with a formulation of a problem area for doctoral dissertation. He has been supported by this grant in the past. Currently, he has been taken off from grant support since he had to leave country temporarily for personal reasons.

Name: Motgi, Nitin**Worked for more than 160 Hours:** Yes**Contribution to Project:**

Nitin been involved in the networking and infrastructure development aspects of the project. He is working on setting up an online compression utility webpage as a test bench for various compnression algorithms and is also working on compressed data transmission infrasturcture tools. Nitin is also working on the development of new lossless compression algorithms for text. He has been supported in this research grant since Fall of 2000.

Name: Awan, Fauzia**Worked for more than 160 Hours:** Yes**Contribution to Project:**

Ms. Fauzia Awan was a student in the gaduate level Multimedia Data Compression course that I taught Spring of 2000 and did a term project related to this project. Since then she got intereted doing a MS thesis under this project and has been working as a Reserach Assistant in the prtoject for one year. She is scheduled to defend her thesis this summer (summer of 2001).

Undergraduate Student

Research Experience for Undergraduates

Organizational Partners

Other Collaborators or Contacts

I have been in touch with two well-known researchers in the data compression field: Tim Bell of Computer Science Department, University of Canterbury, New Zealand and Don Adjeroh of Department of Computer Science and Electrical Engineering, West Virginia University. We have been working on a joint survey paper on 'Pattern Matching in Compressed Text and Images'. This survey has just been finished and I acknowledge the partial support from this grant. This topic is of direct interest to our project and might lead to new research grant proposals to be submitted soon. Also, we are discussing the possibility of linking up our online compression utility website vlsi.cs.ucf.edu with the Canterbury website.

Activities and Findings

Project Activities and Findings: (See PDF version submitted by PI at the end of the report)

Project Summary

The goal of this research project is to develop new lossless text compression algorithms and software tools to incorporate compression for archival storage and transmission over the Internet. The approach consists of pre-processing the text to exploit the natural redundancy of English language to obtain an intermediate transformed form via the use of a dictionary and then compressing it using existing compression algorithms. Several classical compression algorithms such as Huffman, arithmetic, LZ-family (gzip and compress) as well as some of the recent algorithms such as Bzip2, PPM family, DMC, YBS, DC, RK, PPMonstr and recent versions of Bzip2 are used as the backend compression algorithms. The performance of our transforms in combination with these algorithms are compared with the original set of algorithms, taking into account both compression, computation and storage overhead. Information theoretic explanation of experimental results are given. The impact of the research on the future of information technology is to develop data delivery systems with efficient utilization of communication bandwidth and conservation of archival storage. We also develop infrastructure software for rapid delivery of compressed data over the Internet and an online compression utility website as a test bench for comparing various kinds of compression algorithms. The site (vlsi.cs.ucf.edu) will be linked to a very well known compression website which contains the Canterbury and Calgary text corpus. The experimental research is linked to educational goals by rapid dissemination of results via reports, conference and journal papers and doctoral dissertation and master's thesis, and transferring the research knowledge into the graduate curriculum. The PI also delivered invited talks at universities in U.S. (University of California at Santa Barbara, San Diego, Davis, Santa Cruz, Riverside and Oregon State University) and abroad (Indian Institute of Technology, Kharagpur and Indian Statistical Institute, Kolkata).

Goals and Objectives

The goal of this research project is to develop new lossless text compression algorithms and software tools to incorporate compression for archival storage and transmission over the Internet. Specific objectives for this period were:

- À Development of new lossless text compression algorithms.
- À Development of software tools to incorporate compression in text transmission over the Internet and on-line compression utility for a compression test bench.
- À Measurement of performance of the algorithms taking into account both compression and communication metrics.
- À Development of a theory to explain the experimental results based on information theoretic approach.

Executive Summary

The basic philosophy of our compression algorithm is to transform the text into some intermediate form, which can be compressed with better efficiency. The transformation is designed to exploit the natural redundancy of the language. We have developed a class of such transformations each giving better compression performance over the previous ones and all of them giving better compression over most of the current and classical compression algorithms (viz. Huffman, Arithmetic and Gzip (based on LZ77), Bzip2 (based on Burrows & Wheeler Transform), the class of PPM (Partial Predicate Match) algorithms (such as PPMD), RK, DC, YBS and PPMonstr). We also measured the execution times needed to produce the pre-processing and its impact on the total execution time. During our first year of this research grant we developed two transforms (Star(*) and LPT) and two variations of LPT called RLPT and SCLPT. During this reporting period, we developed four new transforms called LIPT, ILPT, LIT and NIT, which produce better results in terms of both compression ratio and execution times. The algorithms use a fixed amount of storage overhead in the form of a word dictionary for the particular corpus of interest and must be shared by the sender and receiver of the compressed files. Typical size of dictionary for the English language is about 1 MB and can be downloaded once along with application programs. If the compression algorithms are going to be used over and over again, which is true in all practical applications, the amortized storage overhead is negligibly small. We also develop efficient data structures to expedite access to the dictionaries

and propose memory management techniques using caching for use in the context of the Internet technologies. Realizing that certain on-line algorithms might prefer not to use a pre-assigned dictionary, we have been developing new algorithms to obtain the transforms dynamically with no dictionary, with small dictionaries (7947 words and 10000 words) and studying the effect of the size of the dictionaries on compression performance. We call this family of algorithms M5zip. One other angle of study is to adapt dynamically to domain-specific corpus (viz. biological, physics, computer science, XML documents, html documents). We experimentally measure the performance of our proposed algorithms and compare with all other algorithms using three corpuses: Calgary, Canterbury and Gutenberg corpus. Finally, we develop an information theory based explanation of the performance of our algorithms.

We make the following contributions during this phase of our work:

1. We develop four new lossless reversible text transforms called Length Index Preserving Transform (LIPT), Initial letter preserving transform (ILPT), Number Index transform (NIT), and Letter Index Transform (LIT). We show that our method of building context, by using word length information and denoting the word length and offset by letters of the alphabet has opened up a new approach for transforming text and for exploring the structural information in order to improve compression performance.

2. We measure performance of our new transforms and obtain the following results:

(a) Bzip2 with LIPT shows an improvement of 5.24% over the original Bzip2 –9, PPMD with LIPT shows an improvement of 4.46% over the original PPMD, and Gzip with LIPT shows an improvement of 6.78% over the original Gzip –9. We also compare LIPT with recent related work and prove that LIPT performs better in terms of compression by giving experimental results. LIPT in conjunction with a modification of Bzip2 called YBS gives 8.0% improvement over Bzip2 –9. YBS with LIPT gives 5.01% improvement over original YBS. Another algorithm called PPMonstr, which is a modification of PPM and is claimed to be faster and more efficient, gives 4.42% improvement with LIPT over the original PPMonstr. RK archiver is claimed to be the best and most efficient latest algorithm. RK with LIPT shows 3.54% improvement over the original RK algorithm.

(b) Our results for new lossless reversible text transforms show that Letter Index Transform (LIT) outperforms LIPT, ILPT, and NIT in compression performance. Bzip2 with ILPT gives 6.83% improvement over the original Bzip2. NIT shows the same compression performance with Bzip2 as ILPT. Bzip2 with LIT outperforms ILPT and NIT. It gives 7.47% improvement over the original Bzip2 and PPMD with LIT gives 6.88% improvement over the original PPMD. Bzip2 with LIT outperforms original PPMD in BPC performance and hence we claim that Bzip2 with LIT is a faster method with better BPC than the much acclaimed PPM based approaches. LIT in conjunction with YBS shows 7.47% improvement over the original YBS, 5.84% improvement with RK over original RK, and 7% improvement with PPMonstr over the original PPMonstr.

3. We present timing performance results for LIPT in conjunction with Bzip2, PPMD and Gzip. Compared to the original algorithms, use of LIPT for preprocessing the text results in 1.7912 times slower compression time than original Bzip2, 3.236 times slower than Gzip and PPMD with LIPT is 1.012 times faster than PPMD. For decoding times, Bzip2 with LIPT is 2.31 times slower than original Bzip2, Gzip with LIPT is 6.56 times slower than Gzip and PPMD with LIPT performs almost the same as PPMD.

4. We also layout dictionary organization for encoding using LIPT. We outline our two level index table structures. We show that the access time depends on the number of words in the source text file. We also show that access time also involves one time quicksort of dictionary words upon initial loading of dictionary into memory which takes time depending on the number of words in the dictionary. Due to these factors, the encoding and decoding time using our transforms is higher than the original algorithms. We give memory overhead comparison of LIPT with Bzip2 and PPMD. LIPT uses 880 K memory compared to 6700 K used by Bzip2 and 5100 K +file size used by PPMD.

5. We are working on a family of new lossless text compression algorithms called M5Zip which obtains the transformed version of the text dynamically with no dictionary, with small dictionaries (7947 words and 10000 words). The transformed text is passed through a pipe of BWT transform, inversion frequency vector, run length encoding and arithmetic coding. Our preliminary results indicate that the algorithm achieves 11.65% improvement over Bzip2 and 5.95% improvement over LIPT plus Bzip2. The investigation on this class of algorithms will continue through next year.

6. We give theoretical explanation of why our transforms are improving the compression performance of the algorithms. We derive mathematical relationships based on entropy and pre-compression. We show that compression in conjunction with our transforms is inversely proportional to the product of file size (ratio of transformed file size to the original file size) factor and entropy of the transformed file. Thus we show that better compression in conjunction with any of our transforms is due to combined effect of pre-compression and entropy.

7. We have developed an internet site (vlsi.cs.ucf.edu) as a test bed for all compression algorithms. To use this, one has to simply clique the “online compression utility” and the client could then submit any text file for compression using all the classical compression algorithms, some of the most recent algorithms including Bzip2, PPMD, YBS, RK and PPMonstr and, of course, all the transformed based algorithms that we developed and reported in this report. The site is still under construction and is evolving. One nice feature is that the client

can submit a text file and obtain statistics of all compression algorithms presented in the form of tables and bar charts. The site is being integrated with the Canterbury website.

In the 'Activities Attached File' (36 pages, 9 figures and 26 tables), we present detail descriptions of the transforms (LIPT, ILPT, NIT and LIT) and experimental results with respect to compression performance, speed and memory overhead and theoretical justification of the observed results.

Project Training and Development:

Major Findings

The major findings can be summarized as follows.

1. We develop four new lossless reversible text transforms called Length Index Preserving Transform (LIPT), Initial letter preserving transform (ILPT), Number Index transform (NIT), and Letter Index Transform (LIT). We show that our method for building context, by using word length information and denoting the word length and offset by letters of the alphabet has opened up a new approach for transforming text and for exploring the structural information in order to improve compression performance.

2. We measure performance of our new transforms and obtain the following results:

(a) Bzip2 with LIPT shows an improvement of 5.24% over the original Bzip2, PPMD with LIPT shows an improvement of 4.46% over the original PPMD, and Gzip with LIPT shows an improvement of 6.78% over the original Gzip. We also compare LIPT with recent related work and prove that LIPT performs better in terms of compression by giving experimental results. LIPT in conjunction with a modification of Bzip2 called YBS gives 8.0% improvement over Bzip2. YBS with LIPT gives 5.01% improvement over original YBS. Another algorithm called PPMonstr, which is a modification of PPM and is claimed to be faster and more efficient, gives 4.42% improvement with LIPT over the original PPMonstr. RK archiver is claimed to be the best and most efficient latest algorithm. RK with LIPT shows 3.54% improvement over the original RK algorithm.

(b) Our results for new lossless reversible text transforms show that Letter Index Transform (LIT) outperforms LIPT, ILPT, and NIT in compression performance. Bzip2 with ILPT gives 6.83% improvement over the original Bzip2. NIT shows the same compression performance with Bzip2 as ILPT. Bzip2 with LIT outperforms ILPT and NIT. It gives 7.47% improvement over the original Bzip2 and PPMD with LIT gives 6.88% improvement over the original PPMD. Bzip2 with LIT outperforms original PPMD in BPC performance and hence we claim that Bzip2 with LIT is a faster method with better BPC than the much acclaimed PPM based approaches. LIT in conjunction with YBS shows 7.47% improvement over the original YBS, 5.84% improvement with RK over original RK, and 7% improvement with PPMonstr over the original PPMonstr.

3. We present timing performance results for LIPT in conjunction with Bzip2, PPMD and Gzip. Compared to the original algorithms, use of LIPT for preprocessing the text results in 1.7912 times slower compression time than original Bzip2, 3.236 times slower than GZIP and PPMD with LIPT is 1.012 times faster than PPMD. For decoding times, Bzip2 with LIPT is 2.31 times slower than original Bzip2, Gzip with LIPT is 6.56 times slower than Gzip and PPMD with LIPT performs almost the same as PPMD.

4. We give theoretical explanation of why our transforms are improving the compression performance of the compression algorithms. We derive mathematical relationships based on entropy and pre-compression. We show that compression in conjunction with our transforms is inversely proportional to the product of file size (ratio of transformed file size to the original file size) factor and entropy of the transformed file. Thus we show that better compression in conjunction with any of our transforms is due to combined effect of pre-compression and entropy.

5. We have developed an internet site (vlsi.cs.ucf.edu) as a test bed for all compression algorithms. To use this, one has to simply click the online compression utility; and the client could then submit any text file for compression using all the classical compression algorithms, some of the most recent algorithms including Bzip2, PPMD, YBS, RK and PPMonstr and of course, all the transformed based algorithms that we developed and reported in this report. The site is still under construction and is evolving. One nice feature is that the client can submit one text file and obtain statistics of all compression algorithms presented in the form of tables and bar charts. The site is now being integrated with the Canterbury website.

Research Training:

Four Ph.D. students and four Masters students have participated and contributed in this research project, but not all of them received direct support from the grant. Dr. Robert Franceschini and Mr. Holger Kruse acquired valuable research experience working on this project and making some early contributions. A Masters student Ms. Fauzia Awan has defended her thesis and is scheduled to graduate this summer. One Masters student Mr. Raja Iqbal briefly collaborated with Ms. Awan in her research. Currently, one Ph. D. student (Mr. Nan Zhang) and one Masters Student (Mr. Nitin Motgi) are working on the project. Other members of the M5 Research Group at the School of Electrical Engineering and Computer Science, Dr. Kunal Mukherjee, Mr. Tao Tao, and Mr. Piyush Jamkhandi, made critical comments and observation during the course of this work. All these students have now graduated. Tao Tao has just started to work for his Ph.D. again. The members of this group met every week to discuss reserach problems and make presentations on their work. This gave them experience of teaching graduate level courses and seminars. One member of this group , Dr. Franceschini, is now a faculty member at UCF. The overall effect of these activities is to train graduate students with the current research on the forefront of technology. Each one of them acquired valuable experience in undertaking significant programming tasks.

Outreach Activities:**Journal Publications**

Tim Bell, Don Adjeroh and Amar Mukherjee, "Pattern Matching in Compressed Text and Images", ACM Computing Survey, p. , vol. , (). Submitted

F. Awan and Amar Mukherjee, "LIPT: A Lossless Text Transform to Improve Compression", Proceedinds of the International Conference on Information Technology:Coding and Communication (ITCC2000), p. 452, vol. , (2001). Published

N. Motgi and Amar Mukherjee, "Network Conscious Text Compression System (NCTCSys)", Proceedings of the International Conference on Information Technology:Coding and Computing (ITCC2001), p. 440, vol. , (2001). Published

Fauzia Awan, Ron Zhang, Nitin Motgi,Raja Iqbal and Amar Mukherjee , "LIPT: A Reversible Lossless Text Transform to Improve Compression Performance", Proc. Data Compression Conferemce, p. 311, vol. , (2001). Published

Books or Other One-time Publications**Web/Internet Site****URL(s):**

<http://vlsi.cs.ucf.edu/>

Description:

This site is for the M5 Reserach Group and the VLSI System Reserach Laboratory under the direction of Professor Amar Mukherjee. A pointer from this site leads to a site relevant to this reserach grant. There is also a pointer to our new "online compression utility".

Other Specific Products**Contributions****Contributions within Discipline:**

We expect that our research will impact the future status of information technology by developing data delivery systems with efficient utilization of communication bandwidths and archival storage. We have developed new lossless text compression algorithms that have improved compression ratio over the best known existing compression algorithms which might translate into a reduction of 75% text traffic on the Internet. We have developed an online compression utility software that will allow an user to submit any text file and obtain compression statistics of all the classical and new compression algorithms. The URL for this is: vlsi.cs.ucf.edu. We are developing software tools to include compression in standard Internet protocols.

Contributions to Other Disciplines:

Contributions to Human Resource Development:

So far four Ph.D. students and four Masters students have participated and contributed in this research project, but not all of them received direct support from the grant. Dr. Robert Franceschini and Mr. Holger Kruse made contributions in the project before it was officially funded by NSF. A Masters student Ms. Fauzia Awan made significant contributions and successfully defended her thesis. A Masters student Mr. Raja Iqbal worked on this project for a brief period of time and collaborated with Ms. Awan in her reserach. Currently, one Ph. D. student (Mr. Nan Zhang) and one Masters Student (Mr. Nitin Motgi) are working on the project. A Ph. D. student Mr. Tao Tao who finished his Masters thesis last year will join our reserach team. Other members of the M5 Research Group at the School of Electrical Engineering and Computer Science, Dr. Kunal Mukherjee and Mr.Piyush Jamkhandi, made critical comments and observation during the course of this work. The overall effect of these activities is to train graduate students with the current research on the forefront of technology.

Contributions to Science and Technology Infrastructure:

We have taught (in the spring 2000 semester) a new course entitled 'CAP5937:Multimedia Compression on the Internet'. The course will be taught again spring of 2001 with a new number CAP5015. This has a new URL location: <http://www.cs.ucf.edu/courses/cap5015/>. This is a graduate level course and 14 students enrolled in the Spring 2000 semester. We are expecting about the same number in Spring 2001. This particular topic has grown directly out of the research that we have been conducting for the last couple of years on data compression. Lecture topics have included both text and image compression, including topics from the research on the current NSF grant. The course has now been completely revised for next offering. The PI also delivered invited talks on research supported by this grant and in general on lossles text compression at universities in U.S. (University of California at Santa Barbara, San Diego, Riverside, Santa Cruz and Oregon State University) and abroad (Indian Institute of Technology, Kharagpur and Indian Statistial Institue , Kolkata). The PI also gave a demonstration of his work on data compression and the online compression utility web site at the IDM Workshop, 2001, Ft. Worth, Texas (April 29-30) sponsored by NSF.

Contributions: Beyond Science or Engineering:**Special Requirements**

Special reporting requirements: None

Change in Objectives or Scope: None

Unobligated funds: less than 20 percent of current funds

Animal, Human Subjects, Biohazards: None

Categories for which nothing is reported:

Organizational Partners

Activities and Findings: Any Outreach Activities

Any Book

Any Product

Contributions: To Any Other Disciplines

Contributions: Beyond Science or Engineering

Activities Attached File (36 pages, 9 figures and 26 tables)

In this attachment, we give complete descriptions of the transforms (LIPT, ILPT, NIT and LIT), provide extensive experimental results for compression performance, speed and memory overhead. We compare our results with other compression algorithms and develop a theory to explain the performance of our new algorithms from an information theoretic point of view.

Method of Approach

The basic idea underlying the first transform (Franceschini and Mukherjee.1996) that we invented is to define a unique signature of a word by replacing letters in a word by a special placeholder character (*) and at most two characters of the original word. Given such an encoding, the original word can be retrieved from a dictionary that contains a one-to-one mapping between encoded words and original words. The encoding produces an abundance of * characters in the transformed text making it the most frequently occurring character. We reported several variations of this theme in our first annual report with very encouraging results. During last year, we took a different twist to our mapping approach recognizing that the frequency of occurrence of words in the corpus as well as the predominance of certain lengths of words in English language should be factored into our algorithms. The other new idea that we introduced is to be able to access the words during decoding phase in a random access manner so as to obtain fast decoding. This is achieved by generating the address of the words in the dictionary by using, not numbers, but the letters of the alphabet. We need a maximum of three letters to denote an address and these letters introduce artificial but useful context for the backend algorithms to further exploit the redundancy in the intermediate transformed form of the text.

LIPT:Length-Index Preserving Transform

LIPT encoding scheme 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. To support our point of repetition of length of words in English text we gathered word frequency data according to lengths for the Calgary, Canterbury [<http://corpus.Canterbury.ac.nz>], and Gutenberg Corpus [<http://www.promo.net/pg/>]. The results given in Figure 1 show that most words lie in the range of length 1 to 10. Most words have lengths 2 to 4. The word length and word frequency results provided a basis to build context in the transformed text. We call this Length Index Preserving Transform (LIPT). LIPT can be regarded as the first step of a multi-step compression algorithm such as Bzip2 which includes run length encoding, BWT, move to front encoding, and Huffman coding. LIPT can be used as an additional component in the Bzip2 before run length encoding or simply replace it.

A dictionary D of words in the corpus is partitioned into disjoint dictionaries D_i , each containing words of length i , where $i = 1, 2, \dots, n$. Each dictionary D_i 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 D_i . $D_i[j]$ denotes the j^{th} word in the dictionary D_i . In LIPT, the word $D_i[j]$, in the dictionary D is represented as $* c_{len}[c][c][c]$

(the square brackets denote the optional occurrence of a letter of the alphabet enclosed and are not part of the transformed representation) where c_{len} stands for a letter in the alphabet [a-z, A-Z] each denoting a corresponding length [1-26, 27-52] and each c is in [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 $1 \leq j \leq 52$ the encoding is $*c_{len}c$; for $53 \leq j \leq 2756$ it is $*c_{len}cc$, and for $2757 \leq j \leq 140608$ it is $*c_{len}ccc$. Let us denote the dictionary of words containing the transformed words as D_{LIPT} . Thus, the 0th word of length 10 in the dictionary D will be encoded as “*j” in D_{LIPT} , $D_{10}[1]$ 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.

Our scheme allows for a total of 140608 encodings for each word length. Since the maximum length of English words is around 22 and the maximum number of words in any D_i in our English dictionary is less than 10,000, our scheme covers all English words in our dictionary and leaves enough room for future expansion. If the word in the input text is not in the English dictionary (viz. a new word in the lexicon) it will be passed to the transformed text unaltered.

Encoding steps

1. The words in the input text are searched in the Dictionary D using a two level index search method.
2. If the input text word is found in the dictionary D, its position and block number (i and j of $D_i[j]$) are noted and the corresponding transformation at the same position and length block in DLIPT is looked up. This transformation is then the encoding for the respective input word. If the input word is not found in dictionary D then it is transferred as it is.
3. Once all the input text has been transformed according to above steps 1 and 2, the transformed text is then fed to a compressor (e.g. Bzip2, PPM etc.).

Decoding steps

1. The received encoded text is first decoded using the same compressor as was used at the sending end and the transformed LIPT text is recovered.
2. Then reverse transformation is applied on this 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.

3. The capitalization mask is applied.

Experimental Results

The performance of LIPT is measured using Bzip2 -9 [Burrows and Wheeler, 1994; Chapin, 2000; Larsson,1998; Seward,2000], PPMD (order 5) [Moffat,1990; Cleary, Teahan and Witten,1995; Salomon,2000] and Gzip -9 [Salomon,2000; Witten, Moffat and Bell,1999] as the backend algorithms. Bzip2 and PPMD are considered best performing compression algorithms in the area of lossless data compression these days. Bzip2 is considered most efficient whereas PPM has the best compression ratio but is very slow in execution. Gzip is very fast and has reasonable compression performance and is also commercially available. Our measurements have compression results in terms of average BPC (bits per character). Note these results include some amount of pre-compression because the size of the LIPT text is smaller than the size of the original text file. By average BPC we mean the un-weighted average (simply taking the average of the BPC of all files) over the entire text corpus. The BPC figures are rounded off to two decimal places and the percentage improvement factors are calculated using actual figures, not rounded BPC values.

Test Corpus

The test corpus is shown in Table 1. Note that all the files given in Table 1 are text files. LIPT is a text transform and only gives better compression results with text files. For this reason, we have left the executable, picture, and binary files out of our test corpus.

File size and dictionary size reduction by LIPT (Pre-Compression)

We used SunOS Ultra-5 to run all our programs and to obtain results. LIPT achieves a sort of pre-compression for all the text files.. We are using a 60,000 English dictionary which takes 557,537 bytes. The LIPT dictionary takes only 330,636 bytes compared to *-encoded dictionary which takes the space storage as that of the original dictionary. Figure 2 shows the comparison of actual file sizes and file sizes obtained after applying LIPT and also after *-Encoding, for some of the text files extracted from Calgary, Canterbury, and Project Gutenberg. From Figure 2 it can be seen that LIPT achieves a bit of compression in addition to preprocessing the text before application of any compressor.

Compression Results

We focus our attention to comparing the performance of LIPT using Bzip2 -9, PPMD (order 5) and Gzip -9 as the backend algorithms. We compute the average BPC for LPT with respect to three corpus and combine the three corpus and compute the average BPC for all the text files. The results can be summarized as follows:

- 1) The average BPC using original Bzip2 is 2.28, and using Bzip2 with LIPT gives average BPC of 2.16, a 5.24% improvement (Table 2).
- 2) The average BPC using original PPMD (order 5) is 2.14, and using PPMD with LIPT gives average BPC of 2.04, and overall improvement of 4.46% (Table 3)
- 3) The average BPC using original Gzip-9 is 2.71, and using Gzip-9 with LIPT the average BPC is 2.52, a 6.78% improvement (Table 4)

Figure 3 gives the comparison of BPC using original Bzip2 and PPMD with BPC using these compressors in conjunction with LIPT for a few text files extracted from our test corpus. From Figure 3 it can be seen that Bzip2 with LIPT (second bar in Figure 3) is close to the original PPMD (third bar in Figure 3) in bits per character (BPC). In instances like paper5, paper4, progl, paper2, asyoulik.txt, and alice29.txt, Bzip2 with LIPT is beating the original PPMD in terms of bits per character (BPC). The difference between average BPC for Bzip2 with LIPT (2.16) and original PPMD (2.1384) is only around 0.02 bits i.e. average BPC for Bzip2 with LIPT is only around 1% more than the original PPMD. This observation is important as it contributes towards the efforts being made by different researchers to obtain PPMD BPC performance with a faster compressor. It is shown later on timing results that Bzip2 with LIPT is much faster than the original PPMD. (Note that although Bzip2 with LIPT gives lower BPC than the original Bzip2, the former is much slower than the later as discussed in later in this report). The files in Tables 2,3 and 4 are listed in ascending order of file size. Note that for normal text files, the BPC decreases as the file size increases. This can clearly be seen from the Tables especially part (c) of every table that has three text files from Project Gutenberg. Table 5 gives a summary comparison of BPC for the original Bzip2 -9, PPMD (order 5), Gzip -9, Huffman (character based), word-based Arithmetic coding, and these compressors with Star-Encoding, and LIPT. The data in Table 5 shows that LIPT performs much better over Star-Encoding and original algorithms except for character based Huffman and Gzip -9. Table 5 also shows that Star-encoding (*-encoding) gives a better average BPC performance for character-based Huffman, Gzip, and Bzip2 but gives worse average BPC performance for word-based arithmetic coding and PPMD. This is due to use of the non-English words and special symbols in the text. Let us define the missing rate as the percentage of bytes in a file, which is not in a word of our dictionary. In the current test corpus the average missing rate for files is 25.56%, i.e. this percentage of the bytes is kept as it is or some special characters are added. For the files with better performance the missing rate is 23.42%, while the files with worse performance have an average missing rate of 28.58%. These missing words are transformed as they are and can be regarded as “noise” in the star converted file for further compression. Unlike LIPT, most of the bytes hit (i.e. total number of bytes in words found in the original dictionary) are converted to ‘*’ character in Star-encoding. So the untransformed words have very different context to those generated by transformed words. For a pure text file, for example the dictionary itself, the star dictionary has a BPC of 1.88 and original BPC is 2.63 for PPMD. The improvement is 28.5% in this case. Although the average BPC for Star-encoding is worse than original, for PPMD, there are

16 files that show improved BPC, and 12 files show worse BPC. Therefore the amount of hits (number of words in the input text that are also found in English dictionary D) is an important factor for the final compression ratio. For Character based Huffman, Star-encoding performs better than the original Huffman and LIPT with Huffman. This is because in Star-encoding there are repeated occurrences of the character ‘*’ which gets the highest frequency in the Huffman code book and is thus encoded with lowest number of bits resulting in better compression results than the original and the LIPT files.

Comparison with Recent Improvements of BWT and PPM

We focus our attention on improving the performance using LIPT over Bzip2 (which uses BWT), Gzip and PPM algorithms because Bzip2 and PPM outperform other compression methods and Gzip is commercially available and commonly used. Of these, BWT based approach has proved to be the most efficient and a number of efforts have been made to improve its efficiency. The latest efforts include Balkenhol, Kurtz, and Shtarkov [1999], Seward [2000], Chapin [2000], and Arnavut [2000]. PPM on the other hand gives better compression ratio than BWT but is very slow in execution time. A number of efforts have been made to reduce the time for PPM and also to improve the compression ratio. Sadakane, Okazaki, and Imai [2000] have given a method where they have combined PPM and CTW [Willems, Shtarkov and Tjalkens,1995] to get better compression. Effros [2000] has given a new implementation of PPM* with the complexity of BWT. Tables 6 and 7 give a comparison of compression performance (in terms of BPC) of our proposed transform which shows that LIPT has better BPC for most of the files and it has better average BPC than all the other methods cited. Some data in Table 6 and Table 7 have been taken from the references given in the respective columns.

Comparison with Word-based Huffman

Huffman compression method also needs sharing of the same static dictionary at both the sender and receiver end, as does our method. Canonical Huffman [Witten, Moffat and Bell,1999] method assigns variable length addresses to data using bits and LIPT assigns variable length offset in each length block using letters of alphabet. Due to these similarities we compare the word-based Huffman with LIPT (we used Bzip2 as the compressor). We show that Bzip2 with LIPT outperforms word-based Huffman for text files. Huffman and LIPT both sort the dictionary according to frequency of use of words. Canonical Huffman assigns a variable address to the input word, building a tree of locations of words in the dictionary and assigning 0 or 1 to each branch of the path. LIPT also assigns variable addresses to the words using variable offset characters (last three characters in LIPT) but it also exploits the structural information of the input text by including the length of the word in encoding. LIPT also achieves a pre-compression due to the variable offset scheme. In Huffman, if new text is added, the whole frequency distribution table has to be recomputed as well as the Huffman codes for them.

A typical word-based Huffman model is a zero-order word-based semi-static model [see Witten, Moffat and Bell,1999]. Text is parsed at the first pass of scan to extract zero-order words and non-words as well as their frequency distributions. Words are typically

defined as consecutive characters and non-words are typically defined as punctuation, space and control characters. If an unseen word or non-word occurred, normally some escape symbol is transmitted, and then the string is transmitted as sequence of single characters. Some special type of strings can be considered for special representation, for example, the numerical sequences. To handle arbitrarily large sequence of numbers, one way of encoding is to break them in to smaller pieces e.g. groups of four digits. Word-based models can generate a large number of symbols. For example, in our text corpus with the size of 12918882 bytes, there are 70661 words and 5504 non-words. We can not make sure that these may include all or most of the possible words in a huge database since the various words may be generated by the definition of words here. Canonical Huffman code [Seward, 2000] is selected to encode the words. The main reason for using canonical Huffman code is to provide efficient data structures to deal with huge dictionary generated and for fast decompression so that the retrieval is made faster.

Comparing with word-based Huffman coding, LIPT is a preprocessor to transform the original words, which are predefined in a fixed English dictionary, to an artificial “language”. However, every word is unique and has similar context patterns among the words with same length or have similar offset in the different word blocks. The transformation does not generate any direct statistics for the word frequencies. But it extracts deterministic strings within the word, which are encoded by a shorter code in an orderly manner. In LIPT, the words not in the dictionary are either kept in the original form or just appended at the end with a single special character. So when further compression, such as Gzip, BWT, or PPM is performed, the words in the dictionary and not in the dictionary may still have chance to share local contexts. Table 8 shows the BPC comparison. For LIPT, we extract the strings of characters in the text and build the LIPT dictionary for each file. In contrast to the approach given in Witten, Moffat and Bell[1999], we do not include the words composed of digits and mixture of alphabets and digits as well as other special characters. We try to make a fair comparison, however, word-based Huffman still uses a broader definition of “words”. Comparing the average BPC, the Managing Gigabyte word-based Huffman model has a 2.506 BPC for our text corpus. LIPT with Bzip2 has a BPC of 2.169. The gain is 13.439%. LIPT does not give improvement over word based Huffman for files with mixed text such as source files for programming languages. For files with more English word, LIPT shows consistent gain.

YBS, RK, and PPMonstr and LIPT

Now let us compare experimental results for some new approaches based on Bzip2 and PPM in conjunction with LIPT with the original Bzip2 and PPM. Note that the results are only given for Calgary Corpus. YBS [<http://artest1.tripod.com/texts18.html>] is a modification of Bzip2. It uses distance coding instead of move to front (MTF) in Bzip2. Table 9 gives the results for YBS and YBS with LIPT. YBS shows 5.12% improvement over original bzip2 -9 and YBS with LIPT shows 10.28% improvement over original bzip2 -9. YBS with LIPT shows 5.68% improvement over original YBS. From Table 9 it can also be verified that YBS with LIPT gives better BPC for all the text files extracted from the Calgary Corpus. YBS with LIPT has the lowest average BPC for Calgary Corpus and hence is the best in terms of compression performance compared to Bzip2, Bzip2 with LIPT, and original YBS.

The next method we are giving results for is RK [<http://rksoft.virtualave.net/>, <http://www.geocities.com/SiliconValley/Lakes/1401/compress.html>]. Table 10 gives the comparison of BPC. Note that RK with LIPT gives better BPC for almost all the files (except two for which RK performs better). We use optimization options (-mx3 -M10) for RK to run our tests. Table 10 outlines the results for Calgary Corpus. RK is an archiver and is achieving a lot of attention in data compression community for its better compression ratios. RK with LIPT shows 3.3% improvement over original RK

Next we compare PPMD (order 5), PPMD with LIPT , PPMonstr (which is a variant of PPMD by Dmitry Shkarin) and PPMonstr with LIPT. PPMonstr with LIPT outperforms original PPMonstr by 4.63% in average BPC. The results are given in Table 11. From these results we can deduce that RK with LIPT gives the best compression performance in terms of BPC. There is not much detail available on the above-mentioned algorithms but these are claimed to be the best lossless English text compressors.

Timing Performance Measurements

The experiments were carried out on 360MHz Ultra Sparc-III Sun Microsystems machine housing SunOS 5.7 Generic_106541-04. The results are shown in Table 12. Average compression time, for our test corpus, using LIPT with Bzip2 -9, Gzip -9, and PPMD is 79.12% slower, 223% slower and 1.2% faster compared to original Bzip2, Gzip and PPMD, respectively. The corresponding results for decompression times are 93.3% slower, 566% slower and 5.9% faster compared to original Bzip2, Gzip and PPMD, respectively. Compression using Bzip2 with LIPT is 92%

faster and decompression is 98% faster than original PPMD (order 5). In our experiments we compare compression times of Bzip2, Gzip and PPMD against Bzip2 with LIPT, Gzip with LIPT and PPMD with LIPT. During the experiments we have used -9 option for Gzip. This option supports for better compression. Compared to the original algorithms, use of LIPT for preprocessing the text results is 1.7912 times slower than Bzip2, 3.236 times slower than Gzip and 1.012 times faster than simple PPMD. The increase in time over standard methods is due to time spent in preprocessing the input file. Gzip uses -9 option to achieve maximum compression therefore in the table we find that the times for compression using Bzip2 are less than Gzip. When maximum compression option is not used, Gzip runs much faster than Bzip2.

Now we move on to discuss decompression time performance. Decompression time for methods using LIPT includes decompression using compression techniques plus reverse transformation time. The results are shown in Table 13.

Dictionary Organization

LIPT uses a static English language dictionary of 59951 words and having a size of around 0.5 MB. LIPT uses transform dictionary of around 0.3 MB. . The transformation process requires two files namely English dictionary, which consist of most frequently used words, and a transform dictionary, which contains corresponding transforms for the

words in English dictionary. There is one-to-one mapping of word from English to transform dictionary. The words not found in the dictionary are passed as they are.

To generate the LIPT dictionary (which is done offline), we need the source English dictionary to be sorted on blocks of lengths and words in each block should be sorted according to frequency of their use. On the other hand we need a different organization of dictionary for encoding and decoding procedures (which are done online) in order to achieve efficient timing.

We use binary search which on average needs $\log w$ comparisons where w is the number of words in the English dictionary D . To use binary search, we need to sort the dictionary lexicographically. We sort the blocks once on loading the dictionary into memory using Quicksort. For successive searching the access time is $M \times \log w$, where M is number of words in the input file and w is number of words in dictionary. So the total number of comparison is given as

$$w \log w + M \times \log w$$

As M gets larger the performance is degraded of the transformation when there are large files. For successive searching the access time is $M \log w$, where M is the number of words in the input file. In physical storage, our dictionary structure is based on first level blocking according length and then within each block we sort the words according to their frequency of use. In memory, we organize the dictionary into two levels. In Level 1, we classify the words in dictionary based on the length of the word and sort these blocks in ascending order of frequency of use. Then in level 2, we sort the words within each block of length lexicographically. This sorting is done once upon loading of dictionaries into the memory. It is subjected to resorting only when there is modification to the dictionary like adding or deleting words from dictionary. In order to search a word of length l and starting character as z , the search domain is only confined to a small block of words which have length l and start with z .

Dictionary Management

It is necessary to maintain a version system for different versions of the English dictionaries being used. When words are added or deleted from the English dictionary, the transform dictionary is affected as the transform has an offset part and the offsets for the words change if there is an addition or deletion of words from the respective block of length in the original dictionary. A simple method works well with our existing dictionary system. When new words are added they are added at the end of the respective word length blocks. Adding words at the end has two advantages: previous dictionary word-transform mapping is preserved scalability without distortion is maintained in the dictionary.

Dictionary Overhead

It is important to note that the dictionary is installed with the executable and is not transmitted every time with the encoded files. The only other time it is transmitted is when there is an update or new version release. The size of the dictionary is 0.5MB (uncompressed) and 197KB when compressed with Bzip2. For achieving a break-even or a gain over the total bits transmitted using the original compression algorithm, the

number of bits transferred using a compression method with LIPT have to be equal or lesser than the bits transferred using the compression method without LIPT. Here we consider the worst case where the dictionary (197 KB -compressed using Bzip2) is also being sent with the compressor and LIPT compressed file. So the total bits being transferred are the bits for the compressed file and the bits for the compressed dictionary. Assume that the uncompressed cumulative total size of the files to be transmitted is F and the uncompressed dictionary size is S_D . The average BPC for compressing a file using Bzip2 with LIPT is 2.16 (all the files in all corpuses combined), and for compressing a file using Bzip2 only it is 2.28. So to get compressed size for the file we need to multiply the average BPC using the respective method by the file size and for the dictionary we need to multiply the dictionary size (in bytes) by the average BPC for Bzip2 as we are using Bzip2 to compress the dictionary. Then for Bzip2 with, we can derive: $F \times 2.16 + S_D \times 2.28 \leq F \times 2.28$. This gives $F \geq 9.5$ MB by replacing S_D with 0.5MB which means that to break even the overhead associated with dictionary, transmission of 9.5MB cumulative data has to be achieved. So if the normal file size for a transmission is say 1 MB then the dictionary overhead will break even after about 9.5 transmissions. All the transmission above this number contributes towards gain achieved by LIPT. Similarly if we use PPMD with LIPT to compress the file and PPMD only for compressing the dictionary: $F \times 2.04 + S_D \times 2.14 \leq F \times 2.14$. This gives $F \geq 10.87$ MB. For Gzip we have $F \times 2.52 + S_D \times 2.71 \leq F \times 2.71$. This yields $F \geq 7.13$ MB. With increasing dictionary size, these thresholds will go up, but in a scenario where thousands of files are transmitted, the amortized cost will be negligible.

Memory Usage

LIPT encoding needs to load original English dictionary (currently 55K bytes) and LIPT dictionary D_{LIPT} (currently 33K). There is an additional overhead of 1.5 K for the two level index tables we are using in our dictionary organization in memory. So currently, LIPT uses about 89K bytes. Bzip2 is claimed to use $400K + (7 \times \text{Block size})$ for compression [<http://krypton.mnsu.edu/krypton/software/bzip2.html>]. We use “-9 option” for Bzip2 and Bzip2 -9 uses 900K of block size for the test. So, we need a total of about 6700K for Bzip2. For decompression it takes around 4600K and 2305K with -s option. For PPMD it takes as about 5100K + file size (this is the size we fix in the source code for PPMD). So, LIPT takes insignificant overhead compared to Bzip2 and PPM in memory usage.

Three New Transforms – ILPT, NIT and LIT

We will briefly describe our attempts at modifying LIPT and we will also present three new lossless reversible text transforms. We will also give experimental results for the new transforms and discuss them briefly. Note that there is no significant effect on the time performance as the dictionary loading method remains the same and the number of words also remain the same in the static English dictionary D and transform dictionaries. Hence we will only give the BPC results obtained with different approaches for the corpus.

Initial Letter Preserving Transform (ILPT)

Initial Letter Preserving transform (ILPT) is similar to LIPT except that we sort the dictionary into blocks based on the lexicographic order of starting letter of the words. We sort the words in each letter block according to descending order of frequency of use. The character denoting length in LIPT (character after ‘*’) is replaced by the starting letter of the input word i.e. instead of $*c_{len}[c][c][c]$, for ILPT it becomes $*c_{init}[c][c][c]$ where c_{init} denotes the initial (starting) letter of the word. Everything else is handled the same way as in LIPT. The results for ILPT in conjunction with Bzip2 are given in Table 14. Bzip2 with ILPT has an average BPC of 2.12 for all the files in all the corpuses combined. This means that Bzip2 -9 with ILPT shows 6.83% improvement over the original Bzip2 -9. Bzip2 -9 with ILPT shows 1.68 % improvement over Bzip2 with LIPT.

Number Index Transform (NIT)

Our encoding scheme uses variable addresses based on letters of alphabet instead of numbers. We wanted to compare this using a simple linear addressing scheme with numbers i.e. giving addresses 0 - 59950 to 59951 words in our dictionary. Using this scheme on our English dictionary D, sorted according to length of words and then sorted according to frequency within each length block, gave deteriorated performance compared to LIPT. So we sorted the dictionary globally according to descending order of word usage frequency. No blocking was used in the new frequency sorted dictionary. The transformed words are still denoted by starting character ‘*’. The first word in the dictionary is encoded as “*0”, the 1000th word is encoded as “*999”, and so on. Special character handling is same as in LIPT. The results are given in Table 15. Note that we compare the BPC results for “Bzip2 -9 with the new transform NIT” with “Bzip2 -9 with LIPT”. Bzip2 with NIT has an average BPC of 2.12 for all the files in all the corpuses combined. This means that Bzip2 -9 with NIT shows 6.83% improvement over the original Bzip2 -9. Bzip2 -9 with NIT shows 1.68 % improvement over Bzip2 with LIPT. The results with NIT confirm the basic principle of data compression. Most frequent words should be represented by least number of characters. This is being done in NIT. The most frequent words are found at the beginning of the dictionary and thus are assigned lower numbers with less number of digits. The interesting observation here is that NIT has almost the same average compression performance as ILPT. This can be observed by the average BPC results. Both have an average BPC of 2.12 for all the files in all three corpuses combined. The results vary for individual files.

Letter Index Transform (LIT)

Now combining the approach taken in NIT and the theory behind using letters to denote offset , we arrive at another transform which is same as NIT except that now we use letters of the alphabet [a-zA-Z] to denote the index or the linear address of the words in the dictionary, instead of numbers. Table 16 gives the results for this encoding scheme called Letter Index Transform (LIT) in conjunction with Bzip2 -9. Note that we compare the BPC results for “Bzip2 -9 with the new transform LIT” with “Bzip2 -9 with LIPT”.

Bzip2 with LIT has an average BPC of 2.11 for all the files in all corpuses combined. This means that Bzip2 –9 with LIT shows 7.47% improvement over the original Bzip2 –9. Bzip2 –9 with LIT shows 2.36 % improvement over Bzip2 with LIPT. The transform dictionary sizes are given in Table 17. Note that LIT dictionary has the smallest size and NIT has the largest dictionary size. From Table 17 it can be seen that although NIT has larger dictionary size but results given in Figures 4, 5 and 6 show that its performance is comparable to that of ILPT. We also note that LIT shows uniform improvement over the other transforms for almost all the files.

Experimental Results for PPMD with LIT

LIT outperforms all other transforms in terms of BPC. Because of its better performance as compared to other transforms, we give experimental results for PPMD with LIPT, and PPMD with LIT in Table 18. PPMD (order 5) with LIT has an average BPC of 1.99 for all the files in all corpuses combined. This means that PPMD (order 5) with LIT shows 6.88% improvement over the original PPMD (order 5). PPMD with LIT shows 2.53 % improvement over PPMD with LIPT.

Experimental Results for YBS, RK and PPMonstr with LIT

We have already shown better performance of YBS, RK, and PPMonstr algorithms when used in conjunction with LIPT. Table 19 gives results of combining LIT with YBS. Results for RK, RK with LIPT in comparison with LIT are given in Table 20. Note that RK with LIT has average BPC lower than RK with LIPT, as expected. Comparison of PPMonstr , PPMonstr with LIPT, and PPMonstr with LIT is shown in Table 21 . Note that the average BPC of PPMonstr with LIT is 2.00, which is same as the average BPC of RK with LIT. From these results, we can see that PPMonstr with LIT, and RK with LIT give the best compression performance in terms of BPC.

Results given above show that LIT performs well with YBS, RK, and PPMonstr giving 7.47%, 5.84%, and 7.0% improvement over the original methods, respectively.

Theoretical Explanation

In this section we give factors behind compression performance. We first give qualitative explanation for better compression with LIPT based on frequency and context. Then we give theoretical explanation based on entropy calculations based on first order entropy and on higher order context-based entropy. We give mathematical equations to giving factors that affects compression performance. We support our claims with the help of experimental results in conjunction with these equations.

Qualitative Explanation for Better Compression with LIPT

LIPT introduces frequent occurrences of common characters for BWT and good context for PPM as well as it compresses the original text. Cleary, Teahan, and Witten [1995], and Larsson [1998] have discussed the similarity between PPM and Bzip2. PPM uses a probabilistic model based on the context depth and uses the context information

explicitly. On the other hand the frequency of similar patterns and local context affect the performance of BWT implicitly. Fenwick [1996] also explains how BWT exploits the structure in the input text. LIPT introduces added structure along with smaller file size leading to better compression after applying Bzip2 or PPMD.

The offset characters in LIPT represent a variable-length encoding similar to Huffman encoding and produce some initial compression of the text but the difference from Huffman encoding is significant. The address of the word in the dictionary is generated at the modeling rather than entropy encoding level and LIPT exploits the distribution of words in English language based on the length of the words as given in Figure 1. The sequence of letters to denote the address also has some inherent context depending on how many words are in a single group, which also opens another opportunity to be exploited by the backend algorithm at the entropy level.

There are repeated occurrences of words with same length in a usual text file. This factor contributes in introducing good and frequent context and thus higher probability of occurrence of same characters (space, '*', and characters denoting length of words) that enhance the performance of Bzip2 (which uses BWT) and PPM as proved by results given earlier in the report. LIPT generates encoded file, which is smaller in size than the original text file. This smaller file is fed to the compression algorithm. Because of the small input file along with a set of artificial but well defined deterministic context, both BWT and PPM can exploit the context information very effectively producing a compressed file that is smaller than the file without using LIPT. In order to verify our conjecture that LIPT may produce effective context information based on the frequent word length recurrence in the text, we made some measurement on order statistics of the file entropy with and without using LIPT and calculated the effective BPC over context length up to 5. The results are given in Table 22 with respect to a typical file `alice29.txt` for the purpose of illustration.

The data in Table 22 shows that LIPT uses less number of bits for context order 3, 2, and 1 as compared to the original file. The reason for this can be derived from our discussion on frequency of recurrence of length throughout the input text. For instance in `alice29.txt` there are 6184 words of length 3 and 5357 words of length 4 (there are words of other lengths as well but here we are taking lengths 3 and 4 to illustrate our point). Out of these, 5445 words of length 3 and 4066 words of length 4 are found in our English dictionary. This means that the sequence, space followed by '*c', will be found 5445 times and the sequence, space followed by '*d', will be found 4066 times in the transformed text for `alice29.txt` using LIPT. There can be any sequence of offset letters after 'c' or 'd' but at least these three characters (including space) will be found in this sequence this many times. Here these three characters define a context of length three. Similarly there will be other repeated lengths in the text. The sequence space followed by '*c' with two characters defines context of length two that is found in the transformed text very frequently. We can see from Table 22 that the BPC for context length 2 and 3 is much lower than the rest. Apart from the space, '*', and the length character sequence the offset letters also provide added probability for finding similar contexts. LIPT achieves a pre-compression at the transformation stage (before actual compression). This is due to the fact that transformation for a word can use at most 6 characters. So words of length 7 and above are encoded with fewer characters. Also the words with other lengths can be

encoded with fewer characters depending on the frequency of their usage (their offset will have fewer characters in this case). Due to sorting of dictionary according to frequency and length blocking we have been able to achieve reduction in size of the original files in the range of 7% to 20%.

Entropy Based Explanation for Better Compression Performance of Our Transforms

Let n denote the total number of unique symbols (characters) in a message Q and P_i denote the probability of occurrence of each symbol i . Then entropy S of message Q [Shannon and Weaver, 1938] is given by:

$$S = - \sum_1^n P_i \log_2 P_i \quad (1)$$

Entropy is highest when all the events are equally likely that is when all P_i are $1/n$. The difference between the entropy at this peak (maximum entropy) where probability P is $1/n$ and the actual entropy given by equation (1) is called the redundancy R of the source data.

$$R = -\log_2(1/P) - (- \sum_1^n P_i \log_2 P_i) = -\log_2 n + \sum_1^n P_i \log_2 P_i \quad (2)$$

This implies that when entropy S is equal to maximum entropy $S_{max} = -\log_2 n$ then there is no redundancy and the source text cannot be compressed further. The performance of compression method is measured by the average number of bits it takes to encode a message. This average is called BPC or bits per character. It is calculated by dividing the total number of bits in the encoded message by the total number of characters in the source message. Compression performance can also be measured by compression factor, which is the ratio of the number of characters in the source message to the number of characters in the compressed message.

Explanation Based on First Order (Context Level Zero) Entropy

First order entropy is sum of entropies for each symbol in a text. It is based on context level zero i.e. no prediction based on preceding symbols. Compressed file size can be given in terms of redundancy and original file size. The quantity S/S_{max} is called the relative entropy and gives a measure of fraction of file that can be compressed. The uncompressed file size F and the compressed file size F_c are related as:

$$F_c = F \times \frac{S}{S_{max}}$$

Substituting the value of S/S_{max} from Equation (2), we have

$$F_c = F - \frac{R \times F}{S_{max}} \quad (3)$$

Compression factor C is given by $C = F/ F_c$. For the ASCII character set , S_{max} is 8. Substituting in F_c , we get:

$$C = 8/S \quad (4)$$

This relationship is also verified in Table 23 in compression factor column. Table 23 gives the minimum first-order entropy (Equation 1) and redundancy (Equation 2) data for Calgary Corpus. The maximum entropy is 8 bits/symbol as we are using 256 character set as our alphabet. Table 23 also shows theoretical compression factors for respective files based on minimum first order entropy S and compression factors based on experimental results using Huffman Coding, which is first order entropy coder (context level zero). The last two columns of Table 23 show that the theoretical and experimental compression factors are very close and the compression factor is inversely proportional to redundancy based on first order entropy. Higher compression factor means more compression as compression factor is given by the ratio of uncompressed file size to compressed file size. Redundancy shows how much a file can be compressed. Lower entropy means less number of bits to encode the file resulting in better compression. Table 23 verifies all these observations.

Now let F_t represent the size of the file transformed with any one of our transforms, which has the size F before application of the transform. Remember that our transforms produce a reduced file size for the intermediate text by a factor k , $0 < k < 1$ (except *-transform for which k is slightly greater than 1 implying an expansion). Hence we can write:

$$F_t = F \times k$$

The compressed file size F_c is given by

$$F_c = (F_t \times S_t) / S$$

where S_t is entropy of the transformed file. Intermediate compression factor C_{int} is given by

$$C_{int} = F_t / F_c$$

Substituting the value of F_c in C :

$$C = F / [F_t \times (S_t / 8)]$$

Finally, substituting the value of F_t , we get

$$C = 8 / (k \times S_t) \quad (5)$$

Equation 5 shows that the compression given by *compression factor C is inversely proportional to the product of file size reduction factor k achieved by a transform, and entropy S_t* . Smaller k and proportionally smaller entropy of the transformed file means

higher compression. Equation 5 is same as equation 4 which is for uncompressed (original) files with $k = 1$ and $S_i = S$.

The results for the products $k \times S_i$ for original files and $k \times S_i$ of all our transforms for Calgary corpus are given in Table 24. We also give experimental compression factor C obtained using Huffman Coding (character based). From Table 24 we can see that LIT has the lowest $k \times S_i$ for most of the files and thus will give the best compression performance. The comparison of product of file factor and entropy for all the methods given in Table 24 is also shown in Figure 7. Figure 8 shows the comparison of compression factors. Comparing Figure 7 and Figure 8 we can see that compression increases as the product of file size and entropy decreases. Compression depends on the product of the file size factor k and entropy of the respective file. This is also depicted in Figure 9 which shows that compression is inversely proportional to product of file size factor k and entropy of the respective file S_i (in this example we have taken LIPT so entropy is denoted by S_i).

Explanation based on higher order context-based entropy

A context-based text compression method uses the context of a symbol to predict it. This context has certain length N . This N is called the context order and the method is said to be following order- N Markov Model [Shannon and Weaver, 1938]]. PPMD (order 5) used in experiments in this report uses order-5 Markov Model. Burrows Wheeler method is also a context-based method [Fenwick,1996;Larsson,1998] using Markov Model. Burrows Wheeler method is a context-based compression method but performs a transformation on the input data and then applies a statistical model.

If we define each context level c_i as a finite state of Markov model then the average entropy S_c of a context-based method is given by:

$$S_c = \sum_{i=0}^m P(c_i) S(c_i) \quad (6)$$

where i is the context level and m is the length of maximum context (context order). This average entropy S_c can also be calculated by totaling the bits taken in each context level and dividing it by the total count of characters encoded at that level. Now based on Equation 6, let us give the minimum entropies for a few files in Calgary Corpus. Table 25 gives the comparison of the product of file size reduction k and average context-based entropy S_c for Bzip2, Bzip2 compression factors C_{bzip2} and PPMD compression factors C_{PPMD} for original, LIPT, and LIT files. The maximum context length is 5 (order 5). These calculations are derived from similar information as given in Table 23 for respective files. The files in Table 25 are listed in ascending order of compression factors. From the data given in Table 25 it can be seen that compression factor for bzip2, C_{bzip2} and also for PPMD, C_{PPMD} are inversely proportional to $k \times S_c$. One also notes that LIT has the lowest $k \times S_c$ but highest compression factors. PPMD with LIT has the highest compression factors. Hence lower $k \times S_c$ justifies the better compression performance for LIT. The same argument can be applied to other transforms.

Now let us compare the context level entropies of alice29.txt (Canterbury Corpus) transformed with LIPT and LIT. Table 26 gives level-wise entropies (column 3 and 5). The first column of Table 26 gives the order of the context (context level). The second and the fourth columns give the total count of predicting each input character using the respective context order for respective methods. For instance the count 108186 for LIPT in the row for context level 5 denotes that 108186 characters were predicted using up to five preceding characters. The third and fifth columns give the entropies or the bits needed to encode each character in the respective context level. The entropy in each context level is calculated by multiplying the probability of the context level, which is the count for that context level divided by the total count (size of file in bytes), by the total number of bits needed to encode all the characters in the respective context. Bits needed to encode each character in respective context is calculated dynamically based on the probability of occurrence of input character given a certain preceding string of characters with length equal to respective context order. The \log_2 of this probability gives the number of bits to encode that particular character. The sum off all number of bits needed to encode each character using the probability of occurrence of that character after a certain string in the respective context gives the total bits needed to encode all the characters in that context level.

Note that in Table 26 by average (in last row of the table) we mean the weighted average which is the sum of BPC multiplied by respective count in each row, divided by the total count. Note also that the average entropy for PPMD with LIT is more than the entropy for PPMD with LIPT, whereas LIT outperforms LIPT in average BPC performance. Although the entropy is higher but note that the total size of the respective transform files are different. LIPT has larger file size compared to ILPT. Multiplying entropy which is average bits/symbol or character with total count from Table 26, for LIPT we have $2.32 \times 140332 = 325570.24$ bits, which is equal to 40696.28 bytes. From Table 26 for LIT, we have $2.50 \times 122456 = 306140$ bits, which is equal to 38267.5 bytes. We see that the total number of bits and hence bytes for LIT is less than LIPT. This argument can be applied to justify performance of our transforms with Bzip2 as it is BWT based method and BWT exploits context. This can be shown for other transforms and for other files as well. Here we have only given one file as an example to explain. This example can be generalized to other transforms and files. We have obtained this data for all the files in Calgary corpus as well.

Our transforms keep the word level context of the original text file but adopt a new context structure on the character level. The frequency of the repeated words remains the same in both the original and transformed file. The frequency of characters is different. Both these factors along with the reduction in file size contribute towards the better compression with our transforms. The context structure affects the entropy S or S_i and reduction in file size affects k . We have already discussed that compression is inversely proportional to the product of these two variables. In all our transforms, to generate our transformed dictionary, we have sorted the words according to frequency of usage in our English Dictionary D. For LIPT, words in each length block in English Dictionary D are sorted in descending order according to frequency. For ILPT, there is a sorting based on descending order of frequency inside each initial letter block of English Dictionary D. For NIT, there is no blocking of words. The whole dictionary is one block. The words in the dictionary are sorted in descending order of frequency. LIT uses the same structure of

dictionary as NIT. Sorting of words according to frequency plays a vital role in the size of transformed file and also its entropy. Arranging the words in descending order of usage frequency results in shorter codes for more frequently occurring words and larger for less frequently occurring words. This fact leads to smaller file sizes.

List of References

1. Z. Arnavut, "Move-to-Front and Inversion Coding", Proceedings of Data Compression Conference, IEEE Computer Society, Snowbird, Utah, March 2000, pp. 193-202.
2. B. Balkenhol, S. Kurtz, and Y. M. Shtarkov, "Modifications of the Burrows Wheeler Data Society, Snowbird Utah, March 1999, pp. 188-197. Compression Algorithm", Proceedings of Data Compression Conference, IEEE Computer
3. M. Burrows and D.J. Wheeler, "A Block-Sorting Lossless Data Compression Algorithm", SRC Research Report 124, Digital Systems Research Center, Palo Alto, CA, 1994.
4. B. Chapin, "Switching Between Two On-line List Update Algorithms for Higher Compression of Burrows-Wheeler Transformed Data", Proceedings of Data Compression Conference, IEEE Computer Society, Snowbird Utah, March 2000, pp. 183-191.
5. J.G. Cleary, W.J. Teahan, and Ian H. Witten, "Unbounded Length Contexts for PPM", Proceedings of Data Compression Conference, IEEE Computer Society, Snowbird Utah, March 1995, pp. 52-61.
6. M. Effros, "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.
7. P. Fenwick, "Block Sorting Text Compression", Proceedings of the 19th Australian Computer Science Conference, Melbourne, Australia, January 31 – February 2, 1996.
8. R. Franceschini and A. Mukherjee, "Data Compression Using Encrypted Text", Proceedings of the third Forum on Research and Technology, Advances on Digital Libraries, ADL 96, pp. 130-138.
9. N.J. Larsson, "The Context Trees of Block Sorting Compression", Proceedings of Data Compression Conference, IEEE Computer Society, Snowbird Utah, 1998, pp 189-198.
10. A. Moffat, "Implementing the PPM data Compression Scheme", IEEE Transaction on Communications, 38(11), pp.1917-1921, 1990

11. K. Sadakane, T. Okazaki, and H. Imai, "Implementing the Context Tree Weighting Method for Text Compression", Proceedings of Data Compression Conference, IEEE Computer Society, Snowbird Utah, March 2000, pp. 123-132.
12. J. Seward, "On the Performance of BWT Sorting Algorithms", Proceedings of Data Compression Conference, IEEE Computer Society, Snowbird Utah, March 2000, pp. 173-182.
13. F. Willems, Y.M. Shtarkov, and T.J. Tjalkens, "The Context-Tree Weighting Method: Basic Properties", IEEE Transaction on Information Theory, IT-41(3), 1995, pp. 653-664.
14. I.H. Witten, A. Moffat, T. Bell, "Managing Gigabyte, Compressing and Indexing Documents and Images", 2nd Edition, Morgan Kaufmann Publishers, 1999.
15. D. Salomon, "Data Compression: The Complete Reference", 2nd Edition, Springer Verlag, 2000.
16. C.E. Shannon, W. Weaver, "The Mathematical Theory of Communication", University of Illinois Press, 1998.

Figures (9) and Tables(26)

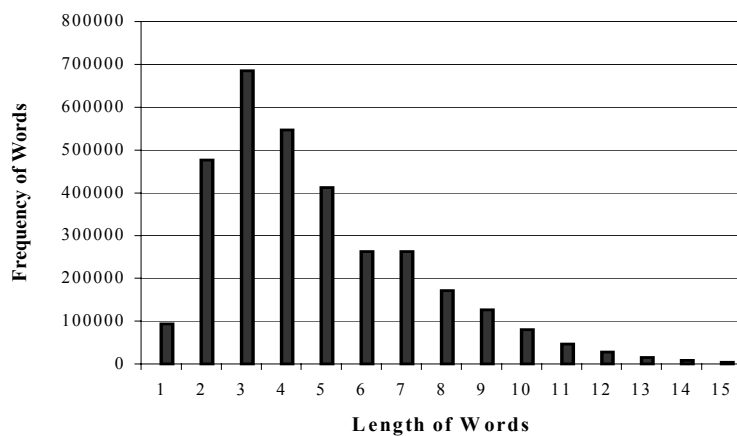


Figure 1. Frequency of English words versus length of words in the test corpus.

File Names	Actual Sizes
Calgary	
Bib	111261
book1	768771
book2	610856
News	377109
paper1	53161
paper2	82199
paper3	46526
paper4	13286
paper5	11954
paper6	38105
Progc	39611
Progl	71646
Progp	49379
Trans	93695

File Names	Actual Sizes
Canterbury	
alice29.txt	152089
asyoulik.txt	125179
cp.html	24603
fields.c	11150
grammar.lsp	3721
lcet10.txt	426754
plrabn12.txt	481861
xargs.l	4227
bible.txt	4047392
kjv.Gutenberg	4846137
world192.txt	2473400
Project Gutenberg	
lmusk10.txt	1344739
anne11.txt	586960
world95.txt	2988578

Table 1: Text files used in our tests (a) Table showing text files and their sizes from Calgary Corpus (b) Table showing text files and their sizes from Canterbury Corpus and Project Gutenberg

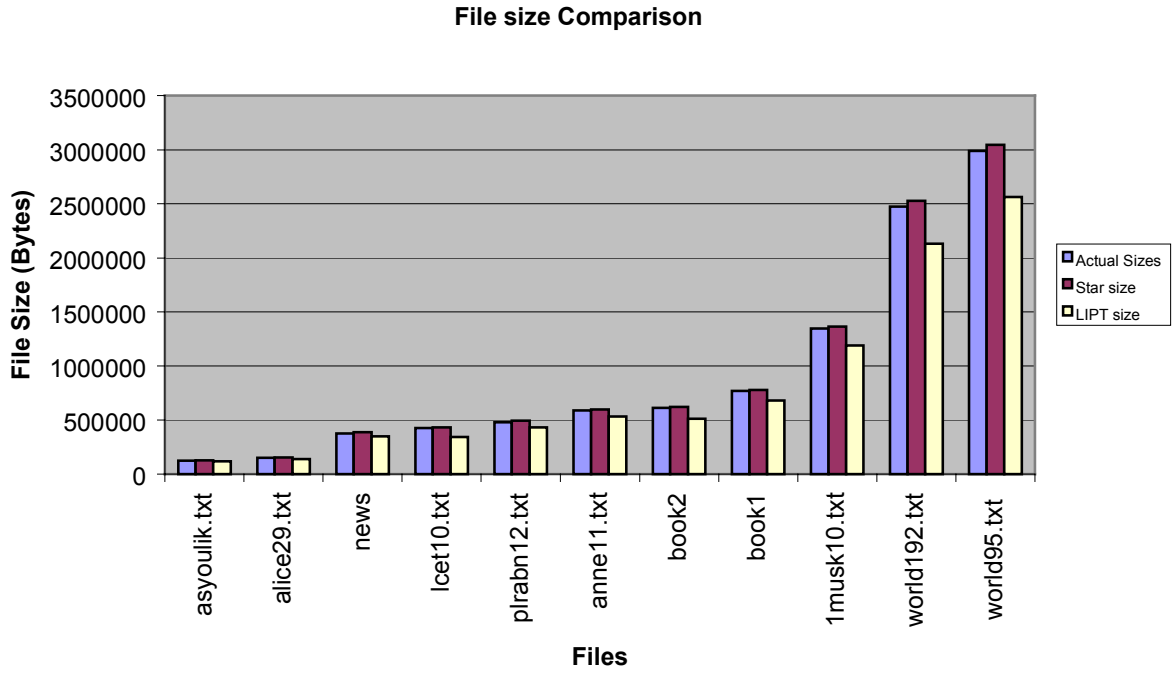


Figure 2: Chart showing the comparison between actual file sizes Star-Encoding and the LIPT file sizes for

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.l	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						

Table 2: Tables a – c show BPC comparison between original Bzip2 –9, and Bzip2 –9 with LIPT for the files in three corpuses

(a)

(b)

(c)

(a)			(b)			(c)		
FileNames	PPMD (BPC)	PPMD with LIPT	FileNames	PPMD (BPC)	PPMD with LIPT (BPC)	FileNames	PPMD (BPC)	PPMD with LIPT
Calgary			Canterbury			Gutenberg		
paper5	2.98	2.74	grammar.lsp	2.36	2.21	Annel1.txt	2.13	2.04
paper4	2.89	2.57	xargs.l	2.94	2.73	lmusk10.txt	1.91	1.85
paper6	2.41	2.29	fields.c	2.04	1.97	World95.txt	1.48	1.45
Progc	2.36	2.30	cp.html	2.26	2.22	Average BPC	1.84	1.78
paper3	2.58	2.37	asyoulik.txt	2.47	2.35			
Progp	1.70	1.68	alice29.txt	2.18	2.06			
paper1	2.33	2.21	lcet10.txt	1.93	1.86			
Progl	1.68	1.61	plravn12.txt	2.32	2.27			
paper2	2.32	2.17	world192.txt	1.49	1.45			
Trans	1.47	1.41	bible.txt	1.60	1.57			
Bib	1.86	1.83	kjv.gutenberg	1.57	1.55			
news	2.35	2.31	Average BPC	2.11	2.02			
book2	1.96	1.91						
book1	2.30	2.23						
Average BPC	2.23	2.12						

Table 3: Tables a – c show BPC comparison between original PPMD (order 5), and PPMD (order 5) with LIPT for the files in three corpora.

(a)			(b)			(c)		
FileNames	Gzip (BPC)	Gzip with LIPT (BPC)	FileNames	Gzip (BPC)	Gzip with LIPT (BPC)	FileNames	Gzip (BPC)	Gzip with LIPT (BPC)
Calgary			Canterbury			Gutenberg		
paper5	3.34	3.05	grammar.lsp	2.68	2.61	Annel1.txt	3.02	2.75
paper4	3.33	2.95	xargs.l	3.32	3.13	lmusk10.txt	2.91	2.62
paper6	2.77	2.61	fields.c	2.25	2.21	World95.txt	2.31	2.15
Progc	2.68	2.61	Cp.html	2.60	2.55	Average BPC	2.75	2.51
paper3	3.11	2.76	asyoulik.txt	3.12	2.89			
Progp	1.81	1.81	alice29.txt	2.85	2.60			
paper1	2.79	2.57	lcet10.txt	2.71	2.42			
Progl	1.80	1.74	plravn12.txt	3.23	2.96			
paper2	2.89	2.62	world192.txt	2.33	2.18			
Trans	1.61	1.56	bible.txt	2.33	2.18			
bib	2.51	2.41	kjv.gutenberg	2.34	2.19			
news	3.06	2.93	Average BPC	2.70	2.54			
book2	2.70	2.48						
book1	3.25	2.96						
Average BPC	2.69	2.51						

Table 4: Tables a – c show BPC comparison between original Gzip -9, and Gzip -9 with LIPT for the files in three corpora.

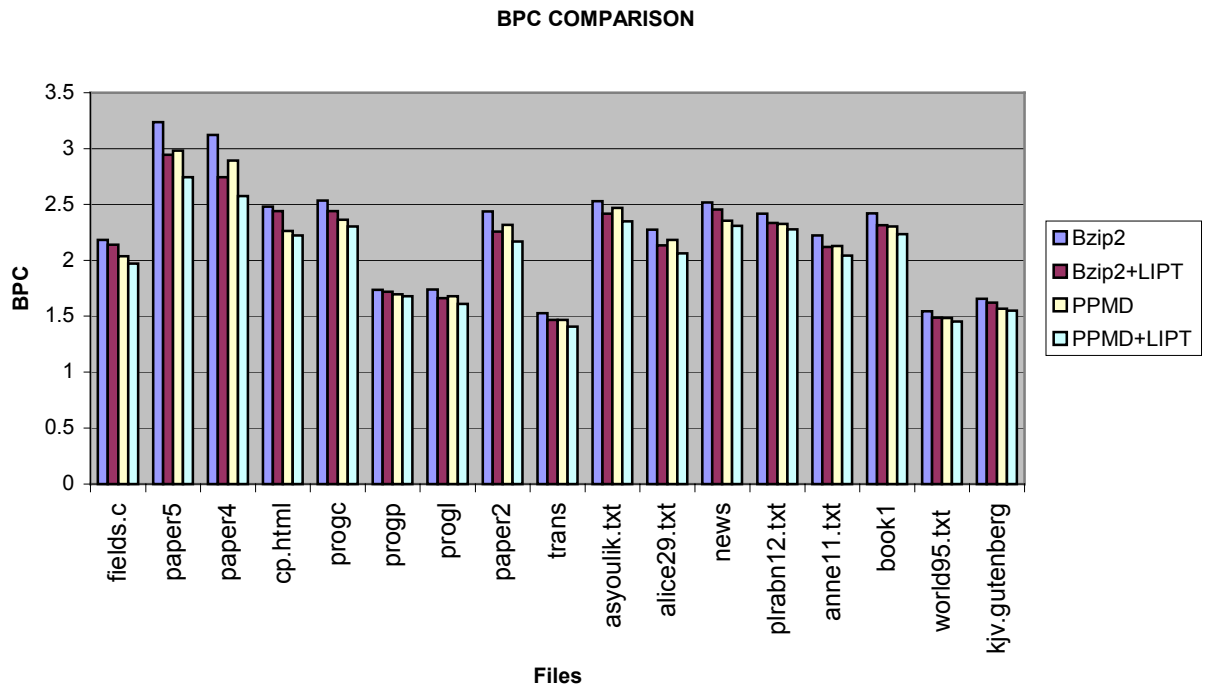


Figure 3: Bar Chart giving comparison of Bzip2, Bzip2 with LIPT, PPMD, and PPPMD with LIPT

	Original (BPC)	*-encoded (BPC)	LIPT (BPC)
Huffman (character based)	4.87	4.12	4.49
Arithmetic (word based)	2.71	2.90	2.61
Gzip-9	2.70	2.52	2.52
Bzip2	2.28	2.24	2.16
PPMD	2.13	2.13	2.04

Table 5: Summary of BPC Results

File	Ref.No.1	Ref. No. 2	Ref. No. 4	Bzip2 with LIPT
bib	2.05	1.94	1.94	1.93
book1	2.29	2.33	2.29	2.31
book2	2.02	2.00	2.00	1.99
news	2.55	2.47	2.48	2.45
paper1	2.59	2.44	2.45	2.33
paper2	2.49	2.39	2.39	2.26
progc	2.68	2.47	2.51	2.44
progl	1.86	1.70	1.71	1.66
progp	1.85	1.69	1.71	1.72
trans	1.63	1.47	1.48	1.47
Average BPC	2.21	2.105	2.11	2.07

Table 6: BPC comparison of approaches based on BWT

File	Ref. No.11	Ref. No. 7	PPMD (order 5) with LIPT
bib	1.86	1.84	1.83
book1	2.22	2.39	2.23
book2	1.92	1.97	1.91
news	2.36	2.37	2.31
paper1	2.33	2.32	2.21
paper2	2.27	2.33	2.17
progc	2.38	2.34	2.30
progl	1.66	1.59	1.61
progp	1.64	1.56	1.68
trans	1.43	1.38	1.41
Average BPC	2.021	2.026	1.98

Table 7: BPC comparison of new approaches based on Prediction Models

	Filesize	Bzip2 with LIPT	Word-based Huffman	% GAIN
cp.html	21333	2.03	2.497	18.696
paper6	32528	2.506	2.740	8.550
progc	33736	2.508	2.567	2.289
paper1	42199	2.554	2.750	7.134
progp	44862	1.733	2.265	23.497
progl	62367	1.729	2.264	23.618
trans	90985	1.368	1.802	24.090
bib	105945	1.642	2.264	27.477
asyoulik.txt	108140	2.525	2.564	1.527
alice29.txt	132534	2.305	2.333	1.194
lcet10.txt	307026	2.466	2.499	1.314
news	324476	2.582	2.966	12.936
book2	479612	2.415	2.840	14.963
anne11.txt	503408	2.377	2.466	3.597
lmusk10.txt	1101083	2.338	2.454	4.726
world192.txt	1884748	1.78	2.600	31.551
world95.txt	2054715	1.83	2.805	34.750
kjv.gutenberg	4116876	1.845	2.275	18.914
AVERAGE		2.169	2.506	13.439

Table 8: BPC comparison for test files using Bzip2 with LIPT, and word-based Huffman

File	Bzip2 (BPC)	Bzip2 with LIPT (BPC)	YBS (BPC)	YBS with LIPT (BPC)
bib.ybs	1.97	1.93	1.93	1.91
book1.ybs	2.42	2.31	2.22	2.16
book2.ybs	2.06	1.99	1.93	1.90
news.ybs	2.52	2.45	2.39	2.36
paper1.ybs	2.49	2.33	2.40	2.28
paper2.ybs	2.44	2.26	2.34	2.18
paper3.ybs	2.72	2.45	2.63	2.39
paper4.ybs	3.12	2.74	3.06	2.71
paper5.ybs	3.24	2.95	3.17	2.91
paper6.ybs	2.58	2.40	2.52	2.36
progc.ybs	2.53	2.44	2.48	2.43
progl.ybs	1.74	1.66	1.69	1.63
progp.ybs	1.74	1.72	1.71	1.70
trans.ybs	1.53	1.47	1.48	1.44
Average	2.36	2.22	2.28	2.17

Table 9: BPC Comparison of Bzip2, Bzip2 with LIPT, YBS, and YBS with LIPT for the text files in Calgary Corpus only

File	Bzip2 (BPC)	Bzip2 with LIPT (BPC)	PPMD (order 5) (BPC)	PPMD with LIPT (BPC)	RK (BPC)	RK with LIPT (BPC)
bib	1.97	1.93	1.86	1.83	1.75	1.72
book1	2.42	2.31	2.30	2.23	2.16	2.16
book2	2.06	1.99	1.96	1.91	1.83	1.85
news	2.52	2.45	2.35	2.31	2.23	2.21
paper1	2.49	2.33	2.33	2.21	2.27	2.17
paper2	2.44	2.26	2.32	2.17	2.21	2.11
paper3	2.72	2.45	2.58	2.37	2.52	2.33
paper4	3.12	2.74	2.89	2.57	2.90	2.63
paper5	3.24	2.95	2.98	2.74	3.01	2.81
paper6	2.58	2.40	2.41	2.29	2.35	2.27
progc	2.53	2.44	2.36	2.30	2.31	2.27
progl	1.74	1.66	1.68	1.61	1.48	1.44
progp	1.74	1.72	1.70	1.68	1.50	1.49
trans	1.53	1.47	1.47	1.41	1.24	1.23
Average	2.36	2.22	2.23	2.12	2.12	2.05

Table 10: BPC Comparison of Bzip2, Bzip2 with LIPT, RK, and RK with LIPT for the text files in Calgary Corpus only

File	PPMD (order 5) (BPC)	PPMD with LIPT (BPC)	PPMonstr (BPC)	PPMonstr with LIPT (BPC)
bib	1.86	1.83	1.83	1.81
book1	2.30	2.23	2.23	2.19
book2	1.96	1.91	1.95	1.91
news	2.35	2.31	2.31	2.28
paper1	2.33	2.21	2.25	2.14
paper2	2.32	2.17	2.21	2.08
paper3	2.58	2.37	2.46	2.25
paper4	2.89	2.57	2.77	2.47
paper5	2.98	2.74	2.86	2.65
paper6	2.41	2.29	2.33	2.23
progc	2.36	2.30	2.29	2.24
progl	1.68	1.61	1.64	1.59
progp	1.70	1.68	1.65	1.64
trans	1.47	1.41	1.45	1.42
Average	2.23	2.12	2.16	2.06

Table 11: BPC Comparison of PPMD, PPMD with LIPT, PPMonstr, and PPMonstr with LIPT for the text files in Calgary Corpus only

Corpus	Average compression time without preprocessing of input file (in sec).			Average compression time when preprocessed using LIPT (in sec).		
	BZIP2	GZIP	PPMD	BZIP2	GZIP	PPMD
Calgary	0.3994	0.4861	12.8256	0.8196	1.4612	12.5968
Canterbury	2.678	3.8393	71.4106	4.5754	11.1488	67.4141
Gutenberg	4.8433	3.4233	103.5167	8.7927	12.4661	105.4828
Total time for whole corpus	7.9207	7.7487	187.7529	14.1877	25.0761	185.4937
Performance of LIPT + {BZIP2, GZIP and PPMD} over simple BZIP2, GZIP and PPMD compression techniques.				1.7912 times slower	3.236 times slower	1.012 times faster

Table 12: Compression times using standard compression methods and compression with LIPT transformation

Corpus	Average decompression time without preprocessing of input file (in sec).			Average decompression time when preprocessed using LIPT reversible transformation. (in sec).		
	BZIP2	GZIP	PPMD	BZIP2	GZIP	PPMD
Calgary	0.1094	0.0233	13.0022	0.1847	0.1058	9.4097
Canterbury	0.6506	0.1166	71.1986	1.1803	0.6977	65.0237
Gutenberg	1.1733	0.2133	100.967	2.3736	1.517	99.7866
Total time for whole corpus	1.9333	0.3532	185.1678	3.7386	2.3205	174.22
Performance of LIPT + {BZIP2, GZIP and PPMD} over simple BZIP2, GZIP and PPMD decompression techniques.				2.31 times slower	6.56 times slower	1.0003 times faster

Table 13: Decompression times using standard decompression methods and decompression with LIPT transformation

FileNames	Bzip2 with LIPT (BPC)	Bzip2 with ILPT (BPC)	FileNames	Bzip2 with LIPT (BPC)	Bzip2 with ILPT (BPC)	FileNames	Bzip2 with LIPT (BPC)	Bzip2 with ILPT (BPC)
Calgary			Canterbury			Gutenberg		
paper5	2.95	2.86	grammar.lsp	2.58	2.54	Anne11.txt	2.12	2.09
paper4	2.74	2.63	xargs.1	3.10	3.02	Imusk10.txt	1.98	1.95
paper6	2.40	2.35	fields.c	2.14	2.12	World95.txt	1.49	1.45
Progc	2.44	2.41	cp.html	2.44	2.40	Average BPC	1.86	1.83
paper3	2.45	2.37	asyoulik.txt	2.42	2.38			
Progp	1.72	1.71	alice29.txt	2.13	2.10			
paper1	2.33	2.26	lcet10.txt	1.91	1.89			
Progl	1.66	1.65	plrabn12.txt	2.33	2.33			
paper2	2.26	2.21	world192.txt	1.52	1.45			
Trans	1.47	1.45	bible.txt	1.62	1.62			
Bib	1.93	1.90	kjv.gutenberg	1.61	1.63			
News	2.45	2.44	Average BPC	2.17	2.13			
Book2	1.99	1.98						
Book1	2.31	2.31						
Average BPC	2.22	2.18						

Table 14: BPC Comparison of Bzip2 with LIPT and Bzip2 with ILPT

FileNames	Bzip2 with LIPT (BPC)	Bzip2 with NIT (BPC)	FileNames	Bzip2 with LIPT (BPC)	Bzip2 with NIT (BPC)	FileNames	Bzip2 with LIPT (BPC)	Bzip2 with NIT (BPC)
Calgary			Canterbury			Gutenberg		
paper5	2.95	2.83	grammar.lsp	2.58	2.51	Anne11.txt	2.12	2.08
paper4	2.74	2.61	xargs.1	3.10	2.99	Imusk10.txt	1.98	1.95
paper6	2.40	2.38	fields.c	2.14	2.13	World95.txt	1.49	1.47
Progc	2.44	2.43	cp.html	2.44	2.37	Average BPC	1.86	1.84
paper3	2.45	2.35	asyoulik.txt	2.42	2.37			
Progp	1.72	1.72	alice29.txt	2.13	2.10			
paper1	2.33	2.28	lcet10.txt	1.91	1.91			
Progl	1.66	1.66	plrabn12.txt	2.33	2.30			
paper2	2.26	2.19	world192.txt	1.52	1.50			
Trans	1.47	1.47	bible.txt	1.62	1.64			
Bib	1.93	1.89	kjv.gutenberg	1.62	1.63			
News	2.45	2.43	Average BPC	2.17	2.13			
Book2	1.99	1.99						
Book1	2.31	2.29						
Average BPC	2.22	2.18						

Table 15: BPC Comparison of Bzip2 with LIPT and Bzip2 with NIT

FileNames	Bzip2 with LIPT (BPC)	Bzip2 with LIT (BPC)	FileNames	Bzip2 with LIPT (BPC)	Bzip2 with LIT (BPC)	FileNames	Bzip2 with LIPT (BPC)	Bzip2 with LIT (BPC)
Calgary			Canterbury			Gutenberg		
paper5	2.95	2.84	grammar.lsp	2.58	2.55	Anne11.txt	2.12	2.08
paper4	2.74	2.60	xargs.1	3.10	2.99	Imusk10.txt	1.98	1.93
paper6	2.40	2.34	fields.c	2.14	2.10	World95.txt	1.49	1.44
Progc	2.44	2.40	cp.html	2.44	2.39	Average BPC	1.86	1.82
paper3	2.45	2.35	asyoulik.txt	2.42	2.37			
Progp	1.72	1.71	alice29.txt	2.13	2.08			
paper1	2.33	2.24	lcet10.txt	1.91	1.88			
Progl	1.66	1.64	plrabn12.txt	2.33	2.31			
paper2	2.26	2.19	world192.txt	1.52	1.46			
Trans	1.47	1.44	bible.txt	1.62	1.60			
Bib	1.93	1.89	kjv.gutenberg	1.62	1.60			
News	2.45	2.42	Average BPC	2.17	2.12			
Book2	1.99	1.96						
Book1	2.31	2.29						
Average BPC	2.22	2.16						

Table 16: BPC Comparison of Bzip2 with LIPT and Bzip2 with LIT

Original	557537 bytes
LIPT	330636 bytes
ILPT	311927 bytes
NIT	408547 bytes
LIT	296947 bytes

Table 17: Dictionary sizes

BPC Comparison of Bzip2 with LIPT,ILPT,LIT, and NIT (for Text files in Calgary Corpus)

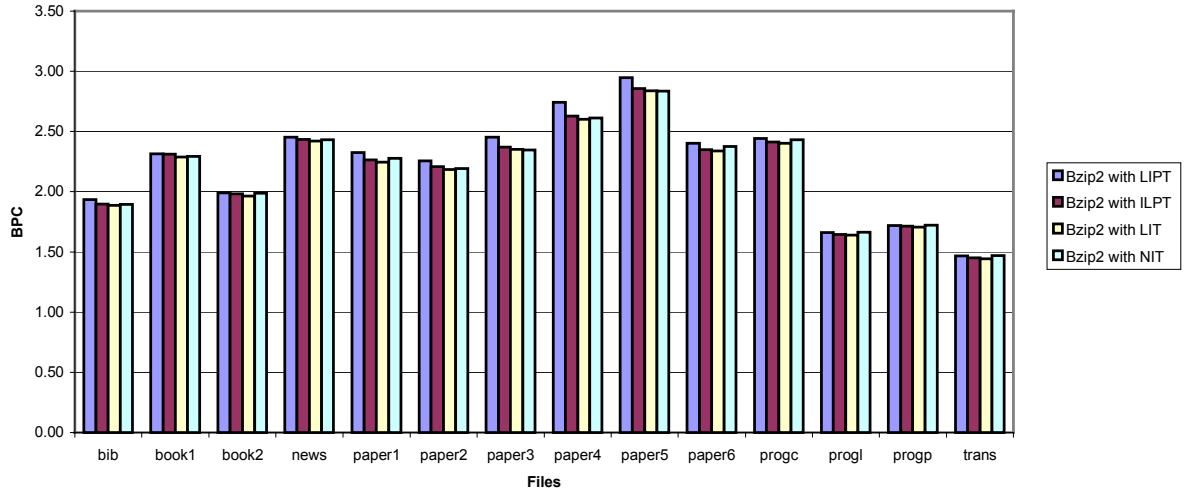


Figure 4: Chart comparing Bzip2 in conjunction with LIPT, ILPT, LIT, and NIT for Calgary corpus

BPC Comparison of Bzip2 with LIPT,ILPT,LIT,and NIT (for Text files in Canterbury Corpus)

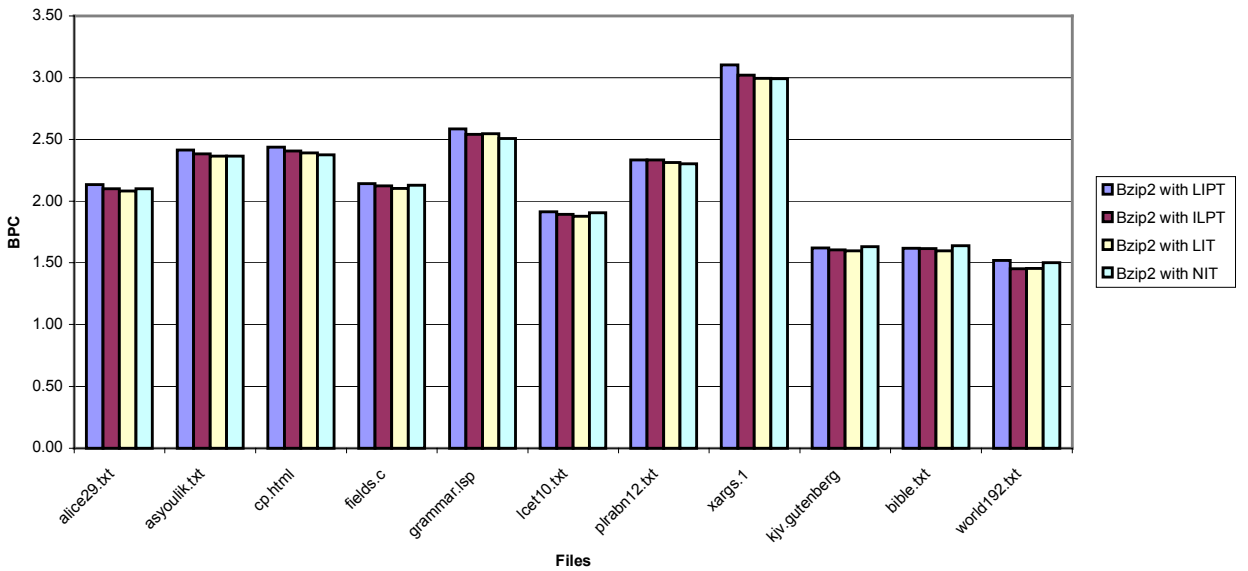


Figure 5: Chart comparing Bzip2 in conjunction with LIPT, ILPT, LIT, and NIT for Canterbury corpus

BPC Comparison of Bzip2 with LIPT,ILPT,LIT,and NIT (for Text files in Gutenberg Corpus)

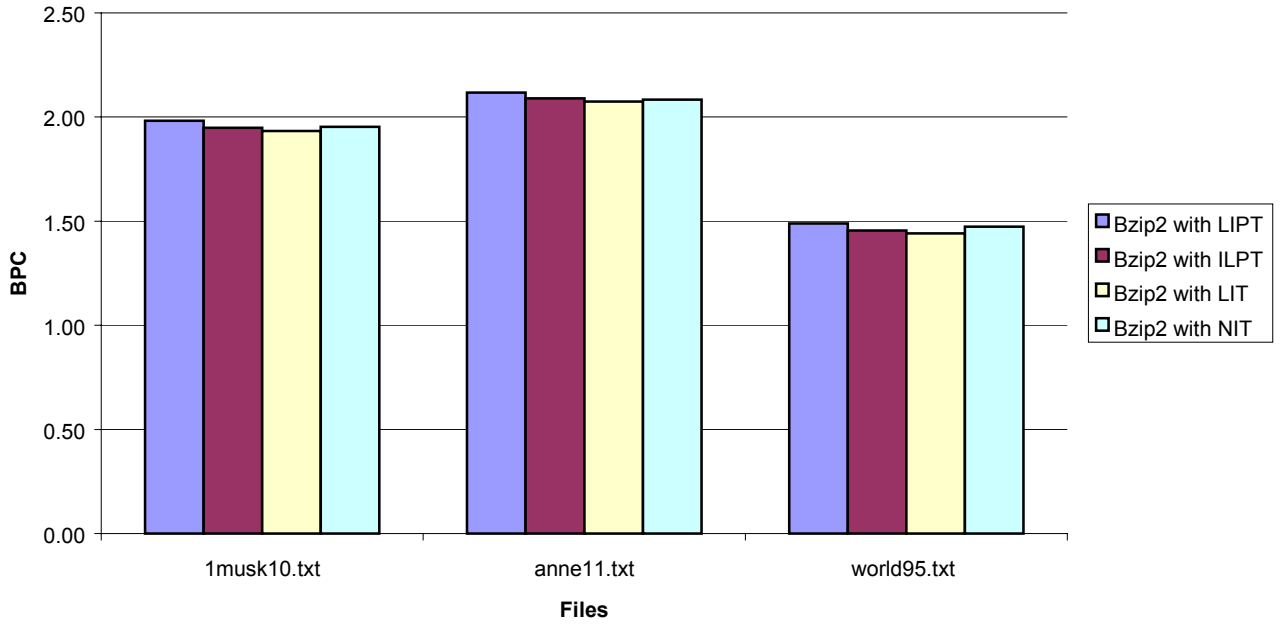


Figure 6: Chart comparing Bzip2 in conjunction with LIPT, ILPT, LIT, NIT, and 4-BIT for Gutenberg corpus

FileNames	PPMD with LIPT	PPMD with 4-BIT	FileNames	PPMD with LIPT	PPMD with 4-BIT	FileNames	PPMD with LIPT	PPMD with LIPT
Calgary			Canterbury			Gutenberg		
paper5	2.74	2.64	grammar.lsp	2.21	2.16	Anne11.txt	2.04	2.00
paper4	2.57	2.46	xargs.1	2.73	2.62	1musk10.txt	1.85	1.82
paper6	2.29	2.21	fields.c	1.97	1.93	World95.txt	1.45	1.38
Progc	2.30	2.25	cp.html	2.22	2.18	Average BPC	1.78	1.74
paper3	2.37	2.28	asyoulik.txt	2.35	2.31			
Progp	1.68	1.66	alice29.txt	2.06	2.01			
paper1	2.21	2.13	lcet10.txt	1.86	1.83			
Progl	1.61	1.56	plrabn12.txt	2.27	2.26			
paper2	2.17	2.10	world192.txt	1.45	1.39			
Trans	1.41	1.37	bible.txt	1.57	1.52			
Bib	1.83	1.79	kjv.gutenberg	1.55	1.51			
news	2.31	2.27	Average BPC	2.02	1.97			
book2	1.91	1.88						
book1	2.23	2.22						
Average BPC	2.12	2.06						

Table 18: BPC comparison of PPMD with LIPT, and PPMD with LIT.

File	YBS (BPC)	YBS with LIPT (BPC)	YBS with LIT (BPC)
bib.ybs	1.93	1.91	1.87
book1.ybs	2.22	2.16	2.14
book2.ybs	1.93	1.90	1.88
news.ybs	2.39	2.36	2.33
paper1.ybs	2.40	2.28	2.21
paper2.ybs	2.34	2.18	2.10
paper3.ybs	2.63	2.39	2.29
paper4.ybs	3.06	2.71	2.58
paper5.ybs	3.17	2.91	2.79
paper6.ybs	2.52	2.36	2.29
prog.ybs	2.48	2.43	2.39
progl.ybs	1.69	1.63	1.61
progp.ybs	1.71	1.70	1.69
trans.ybs	1.48	1.44	1.41
Average	2.28	2.17	2.11

Table 19: BPC comparison of YBS, YBS with LIPT, and YBS with LIT for Calgary Corpus only

File	RK (BPC)	RK with LIPT (BPC)	RK with LIT (BPC)
bib	1.75	1.72	1.70
book1	2.16	2.16	2.15
book2	1.83	1.85	1.84
news	2.23	2.21	2.19
paper1	2.27	2.17	2.11
paper2	2.21	2.11	2.06
paper3	2.52	2.33	2.24
paper4	2.90	2.63	2.49
paper5	3.01	2.81	2.69
paper6	2.35	2.27	2.20
prog	2.31	2.27	2.24
progl	1.48	1.44	1.42
progp	1.50	1.49	1.48
trans	1.24	1.23	1.21
Average	2.12	2.05	2.00

Table 20: BPC comparison of RK, RK with LIPT, and RK with LIT for Calgary Corpus only

File	PPMonstr (BPC)	PPMonstr with LIPT (BPC)	PPMonstr with LIT (BPC)
bib	1.83	1.81	1.77
book1	2.23	2.19	2.14
book2	1.95	1.91	1.86
news	2.31	2.28	2.23
paper1	2.25	2.14	2.06
paper2	2.21	2.08	2.00
paper3	2.46	2.25	2.16
paper4	2.77	2.47	2.35
paper5	2.86	2.65	2.54
paper6	2.33	2.23	2.15
progc	2.29	2.24	2.19
progl	1.64	1.59	1.53
progp	1.65	1.64	1.62
trans	1.45	1.42	1.38
Average	2.16	2.06	2.00

Table 21: BPC Comparison of PPMonstr, PPMonstr with LIPT, and PPMonstr with LIT for the text files in Calgary Corpus only

Context Length (Order)	Original (count)	Original BPC-Entropy	LIPT (count)	LIPT BPC-Entropy
5	114279	1.98	108186	2.09
4	18820	2.50	15266	3.12
3	12306	2.85	7550	2.03
2	5395	3.36	6762	2.18
1	1213	4.10	2489	3.50
0	75	6.20	79	6.35
Total /Average	152088		140332	

Table 22: Comparison of bits per character (BPC) in each context length used by PPM, and PPM with LIPT for the file alice29.txt

File	Minimum first order entropy S (bits/character)	Redundancy R (bits/character)	Compression factor (based on first order entropy S)	Compression factor (based on experimental results using Huffman Coding)
Trans	5.53	2.47	1.45	1.44
Progc	5.20	2.80	1.54	1.53
Bib	5.20	2.80	1.54	1.53
News	5.19	2.81	1.54	1.53
Paper6	5.01	2.99	1.60	1.59
Paper1	4.98	3.02	1.61	1.59
Paper5	4.94	3.06	1.62	1.61
Progp	4.87	3.13	1.64	1.63
book2	4.79	3.21	1.67	1.66
Progl	4.77	3.23	1.68	1.67
Paper4	4.70	3.30	1.70	1.69
Paper3	4.67	3.33	1.71	1.71
Paper2	4.60	3.40	1.74	1.73
book1	4.53	3.47	1.77	1.75

Table 23: Entropy and redundancy in relation with compression factor (Calgary Corpus- original files)

File	Original $k \times S$	C	LIPT $k \times S_l$	C	ILPT $k \times S_l$	C	NIT $k \times S_l$	C	LIT $k \times S_l$	C
Trans	5.53	1.44	5.10	1.56	4.39	1.62	4.61	1.61	4.38	1.64
Bib	5.20	1.53	4.83	1.65	4.63	1.72	4.74	1.68	4.51	1.77
Progc	5.20	1.53	5.12	1.55	2.20	1.61	2.36	1.52	2.13	1.62
News	5.19	1.53	4.96	1.60	4.71	1.69	4.93	1.61	4.60	1.73
Paper6	5.01	1.59	4.58	1.73	4.28	1.85	4.22	1.88	4.11	1.93
Paper1	4.98	1.59	4.42	1.80	4.11	1.93	4.13	1.92	3.99	1.99
Paper5	4.94	1.61	4.41	1.80	4.14	1.92	4.24	1.87	3.98	1.99
Progp	4.87	1.63	4.93	1.61	4.67	1.65	4.94	1.53	4.64	1.66
book2	4.79	1.66	4.20	1.89	3.86	2.05	3.91	2.03	3.71	2.14
Progl	4.77	1.67	4.61	1.72	4.71	1.82	5.09	1.67	4.69	1.83
Paper4	4.70	1.69	4.00	1.98	3.67	2.16	3.70	2.13	3.52	2.25
Paper3	4.67	1.71	3.98	1.99	3.62	2.19	3.74	2.12	3.50	2.26
Paper2	4.60	1.73	4.00	1.98	3.63	2.18	3.70	2.14	3.48	2.28
book1	4.53	1.75	4.05	1.95	3.66	2.17	3.72	2.13	3.50	2.27

Table 24: Product $k \times S$ for original and $k \times S_l$ for all transformed files in Calgary Corpus

Reduction in size and entropy ($k \cdot S$) comparison for Original, LIPT, ILPT, NIT, and LIT files in Calgary Corpus

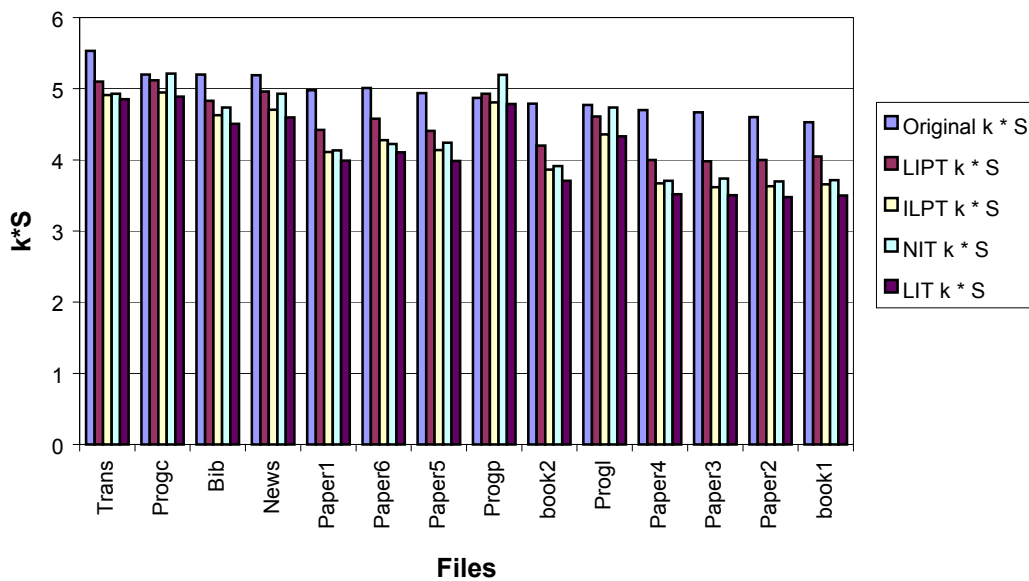


Figure 7 Graph comparing the reduction in size and entropy product for original files, LIPT, ILPT, NIT, and LIT for files in Calgary Corpus.

Comparison of Compression factors for original, LIPT, ILPT, NIT, and LIT files in Calgary Corpus

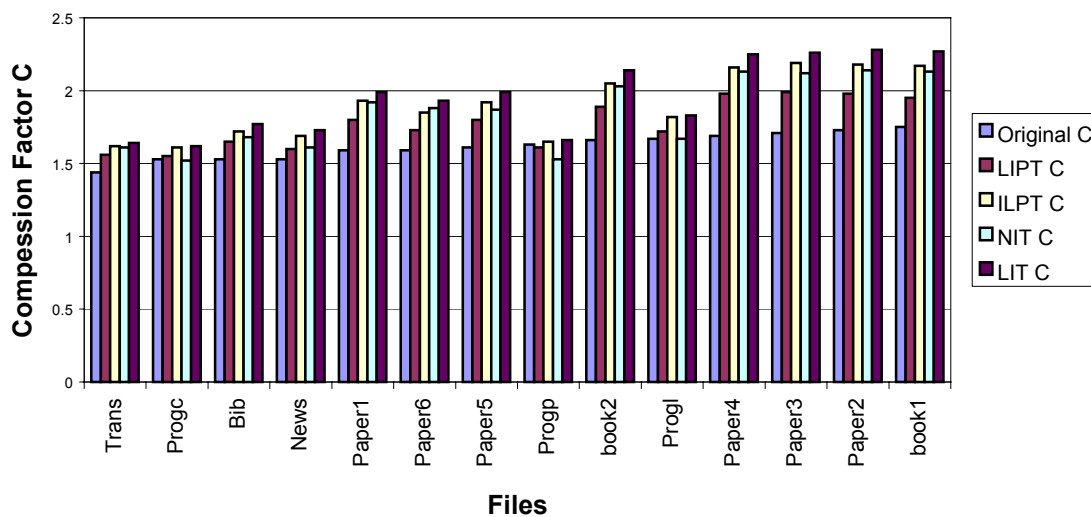


Figure 8 Graph comparing the compression factors using Huffman Coding for original files, LIPT, ILPT, NIT, and LIT for files in Calgary Corpus.

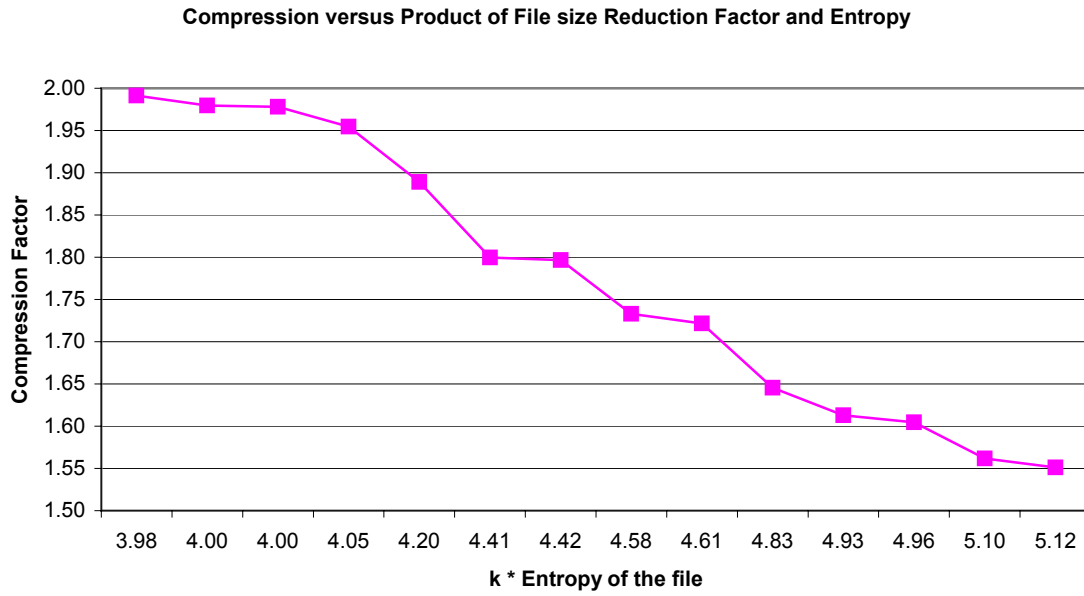


Figure 9: Graph showing Compression versus Product of File size reduction factor k and File Entropy S_c ,

Files	C_{bzip2} (original)	C_{PPMD} (original)	original $k \times S_c$	C_{bzip2} (LIPT)	C_{PPMD} (LIPT)	LIPT $k \times$ S_c	C_{bzip2} (LIT)	C_{PPM} D (LIT)	LIT $k \times$ S_c
paper6	3.10	3.32	2.41	3.33	3.50	2.28	3.42	3.62	2.21
progc	3.16	3.39	2.36	3.28	3.48	2.30	3.33	3.55	2.25
news	3.18	3.40	2.35	3.26	3.47	2.31	3.31	3.52	2.27
book1	3.31	3.47	2.30	3.46	3.58	2.23	3.49	3.61	2.22
book2	3.88	4.07	1.96	4.02	4.19	1.91	4.08	4.26	1.88
bib	4.05	4.30	1.86	4.14	4.37	1.83	4.24	4.47	1.79
trans	4.61	5.45	1.47	5.46	5.69	1.41	5.55	5.84	1.37

Table 25: Relationship between Compression Factors (using Bzip2) and file size and entropy $k \times S_c$ for a few files from Calgary Corpus

Context Length (Order)	LIPT (count)	LIPT (Entropy-BPC)	LIT (count)	LIT (Entropy-BPC)
5	108186	2.09	84534	2.31
4	15266	3.12	18846	3.22
3	7550	2.03	9568	2.90
2	6762	2.18	6703	1.84
1	2489	3.50	2726	3.52
0	79	6.35	79	6.52
Total /Average	140332	2.32	122456	2.50

Table 26: Entropy comparison for LIPT and LIT for alice29.txt from Canterbury Corpus