Semi-Adaptive Substitution Coder for Lossless Text Compression

Rexline S J
Department of Computer Science, Loyola College,
Chennai -600034, India

Robert L
Computer Science and Information System Department, Community College in Al – Qwaiya,
Shaqra University, KSA

Trujilla Lobo
Department of Computer Science, Loyola College,
Chennai -600034, India

ABSTRACT
In this paper, a new text transformation technique called Semi-Adaptive Substitution Coder for Lossless Text Compression is proposed. The rapid advantage of this Substitution Coder is that it substitutes the codewords by referring the reference of the word's position in the dictionary to expedite the dictionary mapping and also codewords are shorter than words and, thus, the same amount of text will require less space. In general, text transformation needs an external dictionary to store the frequently used words. To preserve this transformation method in a healthy way, a semi-adaptive dictionary is used and therefore which reduces the expenditure of memory overhead and speeds up the transformation because of the smaller size dictionary. This new transformation algorithm is implemented and tested using Calgary Corpus and Large Corpus. In this implementation Semi-Adaptive Substitution Coder in connection with a popular bzip2 and commonly used Gzip compressors improve the compression performance by about 7–9% on large files.

Keywords
Transformation, preprocessing, adaptive dictionary, compression, decompression.

1. INTRODUCTION
Lossless text Compression reduces the disk space to store the information and communication costs during transferring the large text file and also the time taken to search the pattern or portion of a file through the huge compressed file [13]. And therefore, Lossless text compression is considered to be an important research area to improve its algorithms and compressing technologies. Compression is possible based on bit level [10], character level and word level compression. Faster compression may also be possible by working with larger units [8].Lossless data compression techniques are often partitioned into statistical based compression techniques and dictionary based compression techniques. Statistical compression algorithm is based on the probability that certain character will occur. Huffman Coding [9] and Arithmetic Coding are the kind of statistical coders.

Dictionary based compression method exploits repetitions in the data. This coding scheme makes use of the fact that certain groups of consecutive characters occur more than once and assign a codeword to that certain occurrences. Most of the dictionary coders are based on LZ77 and LZ78 and are widely employed to compress the data. The main areas of lossless text compression are generation of new compressors or transformation algorithm. Researchers have proved that word based preprocessing method saves the run time memory, boost the compression rates and also speeds up the transmission time [12, 15].Though there are methods which have been existing based on word replacement preprocessing techniques, like Star Encoding [6], LIPT [2] and StarNT [18], it is known that there can be a better word based preprocessing techniques are possible as the days and technologies advancement. The most common preprocessing algorithm used in practice is the dictionary based scheme. These algorithms are based on maintaining a dictionary of words and replacing words of the source file with pointers to identical words in the dictionary [14]. The dictionary can be generated as either static or dynamic manner. There are major methods available for text preprocessing algorithm with static dictionary like Star Encoding, Length-Preserving Transform (LPT), Reverse Length-Preserving Transform (RLPT), Shortened –Context Length-Preserving Transform (SCLPT) Length Index Preserving Transform LIPT [2] and StarNT [18]. The main drawback of the algorithms is that of a fixed initial storage overhead of 1 MB in the form of shared dictionaries [6]. Adaptive dictionary construction method avoids the memory overhead and expedites the dictionary mapping [19], because of the smaller size of the dictionary. In Semi-adaptive dictionary method, an optimal dictionary is prepared during the first pass using the word based distribution in the source file to be compressed and encoding of the source file in to the intermediate file is performed during the second pass by using the dictionary.

The paper is organized as follows: Section 2 presents the existing related text transformation methods; Section 3 proposes the new approach called Semi- Adaptive Substitution Coder for Lossless Text Compression. Section 4 discusses the performance analysis to validate the achievability and efficiency of the proposed method and finally Section 5 contains the conclusions.

2. MATERIALS AND METHODS
The text transformation is a process, which transforms a data into some intermediate form. The transformed data can be compressed with most of existing lossless data compression algorithms, like bzip2, gzip with better compression effectiveness than achieved using an untransformed data. The reverse process is that decompression using given compressor like bzip2, gzip and a reverse preprocessing transformation.

The Burrows-Wheeler transform [11] is a block-sorting lossless data compression algorithm that works by applying a reversible transformation to a block of input data. The BWT can be seen as a sequence of three stages: the initial sorting stage which permutes the input text so similar contexts are grouped together, the Move-To-Front stage which converts the local symbol groups into a single global structure, and the final compression stage which takes advantage of the transformed data to produce efficient compressed output[4]. The transform does not perform any compression but modifies the data in a way to make it easy to compress with a secondary algorithm such as "move-to-front" coding and then Huffman, or arithmetic coding. The BWT algorithm achieves compression performance by using alphabetic reordering as per Chapin [5].

The dictionary based transformation can be performed based on static dictionary or dynamic dictionary creation. Here are major methods available for text preprocessing algorithm with static dictionary like Star Encoding, LIPT [2], StarNT. Skibinski and Grabowski [17] suggested word based replacement transformation method with static dictionary and introduced the recognized preprocessing methods like EOL coding, Capital Conversion, Q-gram replacement, Binary data Filter and Surrounding words with spaces.

Text preprocessing methods with dynamic dictionary are Abel and Teahan preprocessor [1] and the method suggested by Umesh S. Bhadade and A.I. Trivedi [19] .Furthermore ,the transformation can be performed on character or word based method. Umesh S. Bhadade and A.I. Trivedi suggested the method using character based as well as word based dynamic dictionary creation to perform text transformation. In the word based dynamic creation method, they suggested to create four different dictionaries to store the words, prefix and suffix part of the words and non-words which includes non-alphabets that all appear more than once in the source file. In character based dynamic dictionary creation method, they suggested to consider the frequency of all possible 4 character pair, 3 character pair and 2 character pair.

Abel and Teahan [1] proposed five different text transformation algorithms to improve the compression ratio of the source. The recognized transformation algorithms are Capital conversion, EOL coding, word replacement, phase replacement and alphabet reordering. This method needs no external dictionary to maintain the frequently used words and non-words and also this approach is language independent one.

3. PROPOSED CODER

In this section, we proposed our new techniques in text transformation method in which the Semi-Adaptive Coder is designed as two pass preprocessor. During the first pass, frequently used words are extracted from the source file to create an optimal adaptive dictionary and in the second pass only, the source file should be encoded based on the transformation techniques using the adaptive dictionary. The proposed Semi-Adaptive Coder used almost all the most popular well recognized existing transformation techniques like Capital Conversion, EOL coding, N-gram replacement, prefix and word replacement transformation techniques but needs no external storage in the form of static dictionary. So that, this proposed algorithm reduces the cost of maintaining the dictionary and also reduces the transformation time because of the smaller size of the dictionary. If there is not previous knowledge about the source file, then usage of adaptive dictionary is more effective than the static dictionary.

3.1 Semi- Adaptive Dictionary Creation Algorithm

Almost in all the previous transformation techniques, an external static dictionary is used to store the frequently used words [7, 16]. This paper presents the adaptive dictionary in which the lower case version of the words in the source file to be compressed are stored in a table like structure according to its frequency which occurs more than 25 times in the source file and used the indexes of the entries to refer the dictionary words. The shorter codewords are used to represent the index value in the encoded file. When it is analyzed the available codeword, the set of ASCII value 0 to 31 and 127 to 255 that are not used in the text files are determined. Since the ASCII value 0-31 is non-printable characters, it is avoided that characters using as codewords, but used it for flag representation. Remaining characters 127 to 255 are used as codeword for word replacement and N-gram replacement. Here, in this method full word as well as partial words mapping is also permitted in the dictionary [3]. The dictionary is created dynamically in the transformation process and transfers it with the encoded message for decompression.

3.2 Encoding Algorithm

The words in the source file are searched in the Dictionary. If the input text word is found in the dictionary, replace the word with the codeword assigned. Well known recognized transformation techniques like capital letter conversion [6, 17, and 20] also used in this transformation method. Since the proposed method makes use of the capital conversion technique, only the lower case letters of the source file are stored in the dictionary, There are two flags (ASCII values 254 and 255) in the encoded file to indicate that either a given word starts with a capital letter while the following letters are all lowercase, or a given word is of capitals only. Those flags are inserted before the word respectively. Moreover, there is another one flag (ASCII values 253), used to encode lower words with few capital letters. When the word starts with few capital letters and ends with lower case letters, the capital letter flag is placed before the first part of the word and the flag ASCII value 253 is used to separate the lower case part of the word. Furthermore, there is another one flag (ASCII values 6), used for encoding occurrences of flags and the codeword symbol present in the source in the text. If all the letters of a word are in lower case then no flag is present before their codeword. Another one recognized transformation technique that improves the compression performance is End-Of-Line (EOL) coding [1, 17]. The basic scheme is to replace EOL symbols with spaces and a binary flag is used to distinguish an original EOL from a space. It is also necessary to encode information necessary to perform the reverse operation of EOL symbol in a separate stream. N-gram replacement [17] is based on substituting n consecutive characters with single character (ASCII values 227-252). In this method, length of the n does not exceed 3, so that only bigram and trigram characters are replaced with single characters. The remaining codewords ASCII value 127 -226 used for word replacement based on the words present in the adaptive dictionary created during transformation of the source file.

3.3 Decoding Algorithm

The received compressed text is first decompressed using the same compressor as was used at the source end and the transformed text is recovered. The reverse transformation is applied on this decompressed transformed text. If the codeword starts with the ASCII character 128 to 255, then

finds match for words in the dictionary. The transformed codewords are replaced with the respective words in the dictionary. The unaltered word can be easily recognized and transformed as it is in the decoded file. The change of capitalization of the word is performed. It is also necessary to decode the EOL coding based on the information to perform the reverse operation in a separate stream.

4. RESULTS AND DISCUSSION

Semi-Adaptive Substitution Coder for Lossless Text Compression was implemented in C and experiments were carried out on an 800MHz equipped with 3.00 GB RAM, under Windows Vista operating system. To evaluate the performance and excellence of good compression algorithm, there are several criteria to be under consideration such as, the compression ratio and encoding and decoding speed in the case of lossless text compression. In this section to compare the performance of the proposed Coder, the backend algorithms Bzip2 and Gzip are used. The reason to use bzip2 as our backend compressor is that bzip2 compresses files using the Burrows-Wheeler block sorting text compression algorithm and Huffman coding and also bzip2 outperforms other compression algorithms when compared with Gzip, Gzip-9, and DMC by giving the best compression ratios with lowest execution time [6] and Gzip is widely used. The compression ratios are expressed in terms of average BPC (bits per character).

Table 1 Comparative compression ratio of Calgary Corpus with bzip2

File Names	File size Bytes	Bzip2	Encoded file	Sub coder+ Bzip2	% gain
bib	111261	1.97	27283	1.96	0.51
book1	768771	2.42	222177	2.31	4.55
book2	610856	2.06	152744	2.00	2.91
News	377109	2.52	116329	2.47	1.98
paper1	53161	2.46	16397	2.47	-0.41
paper2	8219	2.44	24551	2.39	2.05
Progc	39611	2.53	12490	2.52	0.40
prog1	71646	1.74	15504	1.73	0.57
Progp	49379	1.74	10729	1.74	0
trans	93695	1.53	18039	1.54	-0.65
Average (BPC)		2.14		2.11	1.91

Table 2I Comparative compression ratio of Gutenberg files with bzip2

File Names	File size Bytes	Bzip2	Encoded file	Sub coder+ Bzip2	% gain
World 192	2473400	1.58	452155	1.46	7.59
Bible	4047392	1.67	797929	1.58	5.39
Average (BPC)		1.63		1.52	6.49

Table III Comparative compression ratio of Calgary Corpus with Gzip

File Names	File size Bytes	Gzip	Encoded file	Sub coder+gzip	% gain
bib	111261	2.51	34896	2.51	0.00
book1	768771	3.25	290989	3.03	6.77
book2	610856	2.70	186017	2.44	9.63
News	377109	3.06	137687	2.92	4.58
paper1	53161	2.79	17814	2.68	3.94
paper2	8219	2.89	27660	2.69	6.92
Progc	39611	2.68	13147	2.66	0.75
prog1	71646	1.80	15807	1.77	1.67
Progp	49379	1.80	11111	1.80	0.55
trans	93695	1.61	19156	1.64	-0.19
Average (BPC)		2.51		2.41	3.98

Table 3V Comparative compression ratio of Gutenberg files with Gzip

File Names	File size Bytes	Gzip	Encoded file size	Sub coder+ Gzip	% gain
World192	2473400	2.33	660823	2.14	8.15
Bible	4047392	2.33	1039506	2.05	12.02
Average		2.33		2.10	10.14

The compression performance of the proposed semi-Adaptive Substitution Coder is compared with the results of Bzip2 and Gzip. According to experimental results based on the Calgary Corpus Shown in Table I, the average BPC using original Bzip2 is 2.14. According to Gutenberg files based results taken from R. Franceschini and A. Mukherjee [6] shown in Table II, the average BPC using original Bzip2 is 1.63. But based on the proposed method, average BPC for Calgary Corpus is 2.11, compression gain up to 1.91% and for Gutenberg files is 1.52, compression gain up to 6.49%.

According to experimental results based on the Calgary Corpus Shown in Table III, the average BPC using original gzip is 2.51. According to Gutenberg files based results taken from R. Franceschini and A. Mukherjee [6] shown in Table IV, the average BPC using original gzip is 2.33. But based on this method, average BPC for Calgary Corpus is 2.41, compression gain up to 3.98% and for Gutenberg files are 2.10, compression gain up to 10.14%. When compared with bzip2, gzip gives better improvements in compression ratio.

The results are compared with Abel and Teahan's Universal Text Preprocessing for Data Compression. Their approach is a universal one and also needs no external dictionary. According to this coding technique, a large text file gives better compression than the files with smaller size.

As an example, a section of the text from Canterbury corpus version of world192.txt looks like this in the original text:

For these and other matters, please mail to:

David Turner, Project Gutenberg

Illinois Benedictine College

5700 College Road

Lisle, IL 60532-0900

Running this text through the Semi-adaptive coder yields the following text:

ÿ‰ è¾ ☐ ‘ m”ters, pleËe mail …:

ÿéàµ ÿturner, ÿéáŽ ÿguten¸rg

ÿill*f*o‹ ÿ¸nedict*f*e ÿcollege

5700 ÿcollege ÿéßŒ

ÿl‹le, þil 60532-0900

The speed of encoding is low when compared with decoding time during transformation. Since the costs for the encoding time is in terms of amount of time taken to create the adaptive dictionary and to encode the source file into the intermediate file. Decoding time avoids the creation of dictionary. This proposed method competes with almost the same speed as the existing compressors like bzip2 and Gzip. But it takes additional time during first pass to create the adaptive dictionary

Table V Comparative compression ratio of Abel and Teahan preprocessor and substitution coder

File Names	File size Bytes	Gzip	Abel &Teahan+ Gzip	Sub coder+ Gzip
bib	111261	2.51	2.42	2.51
book1	768771	3.25	3.02	3.03
book2	610856	2.70	2.49	2.44
news	377109	3.06	2.94	2.92
paper1	53161	2.79	2.63	2.68
paper2	8219	2.89	2.65	2.69
progc	39611	2.68	2.61	2.66
prog1	71646	1.80	1.74	1.77
progp	49379	1.80	1.73	1.80
trans	93695	1.61	1.63	1.64
Bible	4047392	2.33	2.11	2.05
World192	2473400	2.33	2.22	2.14

When compared the results with Abel and Teahan's Preprocessor, Larger files give better results compared with smaller files. One reason for not getting better improvements for smaller file may be the fact that there are few word repetitions in the source file. Storing words in a static dictionary is the contrary to our intention. So that words that occurs more than 25 times over the source file used to form a table like dictionary structure gives better performance for large files. Table V shows the comparative compression ratio with Abel and Teahan preprocessor taken from [1]. According to the results, it is proved that large files gives better performance than smaller one.

5. CONCLUSION

The proposed method is admirable extensions of the transformation method experimented on various text files. The most common transformation algorithm used in practice is the dictionary scheme using an external static dictionary. But to avoid the dictionary overhead, adaptive dictionary is proposed to store the frequently used words of the source file. In future work it is worthy to remove the spaces between the words so that the redundancy of space can also be compressed in which smaller files can also be compressed with better performance. Tested Corpus indicated that better compression ratios were possible with large text files. This is an inconclusive research area which leaves lots of space for researchers to develop different preprocessors and new compressors in lossless text compression area.

6. REFERENCES

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

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

[3] T. Bell, J. Cleary, and I. Witten, "Data compression using adaptive coding and partial string matching," IEEE Transactions on Communications, Vol. 32 (4), p. 396-402, 1984.

[4] 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.

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

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

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

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

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

[10] Hussein Al-Bahadili, Shakir M. Hussain," A Bit-level Text Compression Scheme Based on the ACW Algorithm", International Journal of Automation and Computing, pp: 123-131, February 2010.

[11] Isal RYK, Moffat A, Ngai ACH. "*Enhanced Word-Based Block-Sorting Text Compression"*, Proceedings of the 25th Australian Computer Science Conference, Melbourne, pp. 129–138, January 2002.

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

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

[14] Md.Nasim Akhtar,Md.Mamunur Rashid, Md,Shafiqul Islam, Mohammad Abul kashem,Cyrill Y. Kolybanov , "*Position Index preserving Compression for Text Data*",JCS&T,Vol 11, No 1,April 2011.

[15] Radu R,ADESCU, "*Transform Methods Used in Lossless Compression of Text Files* ", romanian journal of information science and technology ,Volume 12, Number 1,pp :101-115, 2009.

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

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

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

[19] Umesh S. Bhadade, A.I. Trivedi, "*Lossless Text Compression using Dictionaries",* International Journal of Computer Applications ,Volume 13– No.8, January 2011.

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