International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:3, No:11, 2009

Dynamic Decompression for Text Files

Ananth Kamath[1], Ankit Kant[2], Aravind Srivatsa[3], Harisha J.A.[4]

[#]*Information Science and Engineering, Rashtreeya Vidyalaya College of Engineering*

[1]ananthrkamath@gmail.com
[2]ankit.k9@gmail.com
[3]aravind.eric@gmail.com
[4]harisha.ja@gmail.com

***Abstract*—**Compression algorithms reduce the redundancy in data representation to decrease the storage required for that data. Lossless compression researchers have developed highly sophisticated approaches, such as Huffman encoding, arithmetic encoding, the Lempel-Ziv (LZ) family, Dynamic Markov Compression (DMC), Prediction by Partial Matching (PPM), and Burrows-Wheeler Transform (BWT) based algorithms. Decompression is also required to retrieve the original data by lossless means. A compression scheme for text files coupled with the principle of dynamic decompression, which decompresses only the section of the compressed text file required by the user instead of decompressing the entire text file. Dynamic decompressed files offer better disk space utilization due to higher compression ratios compared to most of the currently available text file formats.

***Keywords*—**Compression, Dynamic Decompression, Text file format, Portable Document Format, Compression Ratio.

I. Introduction

THE Use of compression for storing text files has become inherent part of personal as well as commercial computing. The various compression applications available perform two functions, compression and decompression. The text document is first compressed and then the entire document is decompressed when required. This has some implications such as the unnecessary use of disk space for storing the compressed document as well as uncompressed document at the same time. Another implication is that even though an end user may require only a part of the document, the entire document as a whole is decompressed.

The algorithm (and application) described in this document addresses both of the above-mentioned problems associated with compression applications and readers. The algorithm performs compression of the text file and displays only the section of the text file required by the user in decompressed format on the developed Graphical User Interface. Therefore it provides a better and efficient way of storing and reading the text files, saving unnecessary wastage of disk space.

II. Related Work

In [1] it has been shown that text file compression can be done by assigning 2 character and 3 character ASCII codes. It has also been shown that about 75% reduction in size is achieved by using it with gzip and bzip2. Also, the number of possible codes is 73680, which is lesser than the number of words in the scheme of compression highlighted in this paper. However, only the compression scheme has been highlighted. [1] does not specify a decompression scheme for dynamically decompressing data.

[9] presents a variable length word-coding scheme, which allows the direct search of the compressed file without decompression of the entire file using a variant of the Boyer Moore algorithm. The compression technique used in [9] provides for efficient decoding of arbitrary portion of text as well as smaller vocabulary representation. In comparison, the compression technique used in this paper uses a much simpler algorithm for compression and searching operations are directly done on the compressed file without decompressing the file either. However, in addition to the features in [9], the compression algorithm used by us allows line wise decoding of the compressed file, which allows dynamic decompression of the file as required by the user.

III. Compression Principle

The Compression algorithm applied assigns a unique 3-character code to every word in the source file. The position of the codes in the compressed file corresponds to the position of the respective words in the source file, i.e. their order of occurrence is not changed. Codes assigned consist of combinations of lowercase and uppercase ASCII alphabets.

Numbers are not compressed and are retained in the compressed file in their original form.

The compression algorithm uses page markers by using a ~ at the start of each page. Each page is determined by assigning a specific number of lines and characters for each line. The page markers play an important role in the current dynamic decompression algorithm as the decompression function uses a page marker count to determine the page number, and then decompresses the required page.

A sequence of spaces is coded by counting the number of spaces and prefixing the two-digit count with #. Newline characters are encoded as the character y. When either a # or y is encountered in the source file, they are prefixed with the character z to ensure that there is no ambiguity while decompressing. One letter and two letter words are coded by prefixing Y and X respectively to them. Words greater than 20 letters in length are not coded either and are prefixed with the character t and suffixed with #. For words with length >2 and <20, every code is in the form of $\alpha\beta\gamma$ where β and γ take values as from a to z and then from A to Z in sequence. The α value is determined from the Table I.

TABLE I
ALPHA VALUES FOR ASSIGNING CODES

Word length	α values	Number of codes
3	a, A	5408
4	b,B	5408
5	c,C	5408
6	d,D	5408
7	e,E	5408
8	f,F	5408
9	g,G	5408
10	h,H	5408
11	i,I	5408
12	j,J	5408
13	k,K	5408
14	l,L	5408
15	m,M	5408
16	n,N	5408
17	o,O	5408
18	p,P	5408
19	q,Q	5408
20	r,R	5408

An example sequence of codes assigned for 3 letter words would be αaa – αaz – αaA - αaZ - αba – αbz αza – αzz – αzA – αzZ – αAa – αAz – αAA – αAZ – αBa – αZa – αZz – αZA – αZZ initially with α = a and after the completion of the sequence with α=A.

The compressed file structure consists of two parts, the codes and the words. The coded form of the source file is stored in the compressed file first. This is followed by a ~ which denotes that the codes have all been stored. For the decompression algorithm to relate each word with its corresponding code, the words are first classified based on their length and then words of a each length are ordered based on the order of their first occurrence in the source file. The words are then stored at the end of the compressed file. Since codes are assigned based on the order of the occurrence, this is a reliable technique for the decompression algorithm to associate each code with its corresponding word.

IV. DYNAMIC DECOMPRESSION

The principle of Dynamic Data Decompression refers to the decompression of a compressed file when it is being viewed. The entire file is not decompressed, but only the page that is currently being viewed is decompressed. The rest of the file remains in the compressed state. Hence better storage efficiency is obtained as the space required on the secondary storage media is reduced by a factor almost equal to the Compression Ratio of the algorithm used for compression.

Given the source file size as S bytes with N pages and the compression ratio as C, the compressed file size = S/C bytes. The average page size in the source file = S/N bytes. The compressed file is then subject to dynamic data decompression. Therefore, average size occupied on disk is

$$(S / C) + (S / N) \text{ bytes} \quad (1)$$

This is a considerable decrease in size from the original S bytes. The dynamic data decompression functionality in its current implementation is provided by a User Interface front end and a decompression function that operates as the back end. When a compressed file is first opened, the first page of it is decompressed and displayed. Command buttons are provided on the user interface to view the next and previous pages. Whenever one of these buttons is clicked on, the User Interface control sends a page number as a parameter to the decompression function. The page specified by the page number is then decompressed by the decompression function, which identifies pages by maintaining a count of page markers encountered. Hence, the page requested only is decompressed. Therefore, at any point of time, the space occupied on disk is the sum of the compressed file size and the decompressed size of the page that is currently being viewed.

A. Decompression Function

The Decompression function is used to decompress the compressed text file. It takes a parameter, the page number of the page to be decompressed and decompresses only this page. It uses a combination of page markers and line markers to distinguish between pages. For every code in the compressed file, the decompression function refers Table 1 to determine the length of the corresponding word for the code. Once the length is determined, it also calculates the displacement from the initial code for words of that length. Based on this displacement value, it determines the corresponding word for the code. This is done by checking the words of the determined length at the end of the compressed file. Since the codes are assigned and words are stored based on the order of occurrence, the displacement value leads the decompression function to the required word.

An added advantage of storing codes is faster searching operations as each word is reduced to a three-character code. Hence, once the code for the word that has to be searched is determined by comparison at the end of the file, the searching operation speed is increased. For Example, even a 15 character word is reduced to a 3 character code and by just comparing the first character of each code, we would be able to determine if it is represents a 15 character word. In any

other case, a comparison that extends till the 15th character is required. But in this case a maximum of 3 comparisons are required.

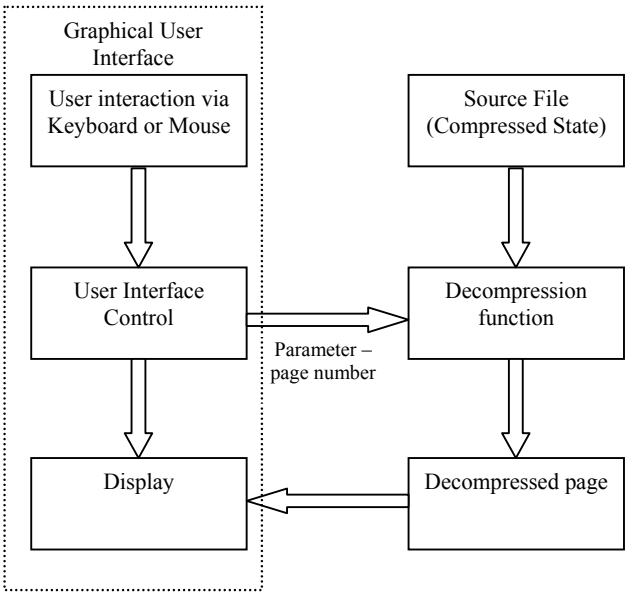

Fig. 1 A Block Diagram showing the interaction between various components of the dynamic decompression system.

V. Experimental Results

For the purpose of comparison with other popular file formats for text file, experiments were conducted. The other file formats used were .docx (Microsoft Word 2007 Document format), .pdf (Adobe Portable Document format), and .txt (Text file format). Every source data was converted into these formats and the size was observed.

Additionally, the source data was compressed using the compression principle that is discussed in this paper. Then, the compressed file was subjected to dynamic decompression and an average of the decompressed page size for a certain range of pages was computed. At any point of time, the size occupied on disk was taken to be the sum of the average decompressed page size and the compressed file size from Equation (1). This value was computed and documented.

Source data of four different sizes were taken and each of them were converted into the different file formats above and also compressed using the principle discussed in this paper and the value for each as per equation (1) were calculated and tabulated in Table II.

TABLE III
Size Comparison for Data in Different File Formats

All values in Kilo bytes (approx.)	Text File Format (.txt)	MS Word Document (.docx)	Adobe PDF (.pdf)	Dynamically Decompressed File
Data 1	48	32	109	39
Data 2	120	60	202	85
Data 3	714	359	1310	509
Data 4	1012	505	1800	668

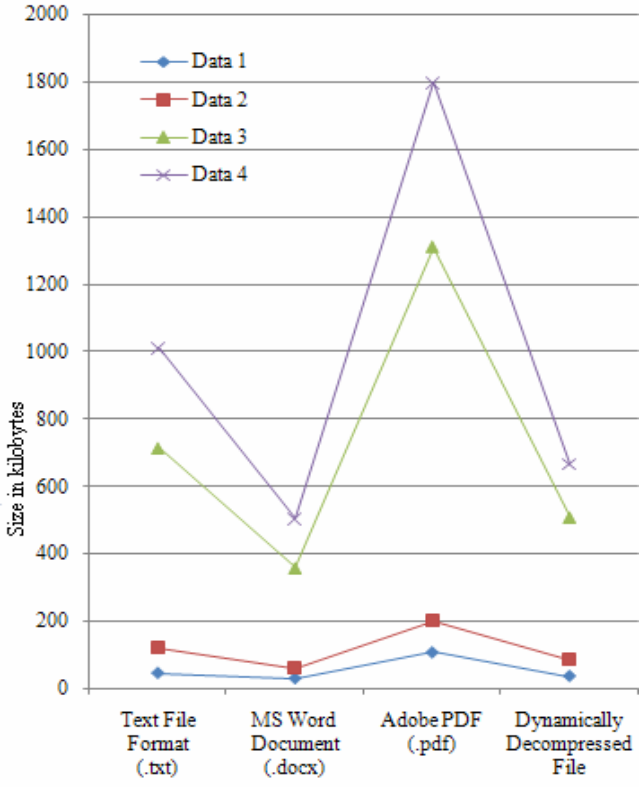

Fig. 2 Graph indicating size of data on disk stored in various file formats

VI. Conclusions

Dynamic decompression reduces the disk space needed to store a text file drastically. For compression applications currently available, there is a point in time wherein both the text file decompressed by the application and the compressed file exist simultaneously on disk. This leads to inefficient disk utilization. The dynamic decompression scheme addresses this anomaly by having only the coded file and only a section of file required by the user decompressed on the disk at any point in time. Therefore, the space occupied on the disk can be reduced significantly. This feature alone makes the developed reader very suitable for low-end consumers as well as in commercial industry where the disk space is of great value

and in comparison to other text file formats currently available, offers much more compression ratio.

Since the compression scheme, even though providing compression more than the most of the available text file formats, doesn't quite compress the text files as effectively as other stand alone compression applications available, there is still scope for improvement in the encoding function that we have used. One way in which it can be achieved is by encoding the words based on their frequency of occurrence in the text file and also doing the encoding at the bit level.

The compression time taken by the application can be reduced by using better data structures available such as a B-Tree, and also by using binary search when searching for a keyword in a list. Since the Graphical User Interface that we have developed currently uses page markers and line markers to decompress pages according to need, we can solve the problem of Graphical User Interface window resizing, wherein the number of characters in a line increase or decrease when resizing a window, by having a character count for a particular window size instead.

A further improvement can be made on the Graphical User Interface front, by providing encryption standards along with the features of dynamic decompression. An editor can also be integrated to the developed reader where in a user can edit the page being viewed and it gets correctly encoded again after he navigates to another page.

ACKNOWLEDGMENT

We would like to thank Professor N.K. Srinath of Information Science and Engineering Department, Rashtreeya Vidyalaya College of Engineering for his inputs and valuable suggestions that resulted in an improved version of our paper.

REFERENCES

[1] 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*.

[2] Behrouz A. Forouzan and Richard F. Gilberg, Computer Science A Structured Programming Approach Using C, Thomson, 2003

[3] Data Structures using C, Aaron M. Tenenbaum, Yedidyah Langsam and Moshe J. Augenstein, Pearson Education, 2006

[4] Michael J. Folk, Bill Zoellick, Greg Ricardi. File Structures-An Object Oriented Approach with C++, Addison-Wesley, 1998

[5] B.S. Shajeemohan and V.K.Govindan, Intelligent Compression Scheme For Faster And Secure Transmission Of Text And Image Data Over Internet, International Conference on Human Machine Interface ICHMI 2004

[6] Marc L. Corliss , E. Christopher Lewis , Amir Roth, The implementation and evaluation of dynamic code decompression using DISE, *ACM Transactions on Embedded Computing Systems (TECS*), v.4 n.1, p.38-72, February 2005.

[7] 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.

[8] U. Manber, "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.

[9] "A Scheme That Facilitates Searching And Partial Decompression Of Textual Documents. Ashutosh Gupta . *Intl. Journal of Advanced Computer Engineering*, Volume 1, No 2, pages 99 -109, 2008.

[10] F. Awan, N. Zhang, N. Motgi, R. Iqbal, and A. Mukherjee, "LIPT: A Reversible Lossless Text Transform to Improve Compression Performance," *Proceedings IEEE Data* Compression Conference, pp. 481-210, 2001.

[11] D. A. Huffman, "A method for the construction of minimum redundancy codes," *In Proc. IRE 40*, volume 10, pages 1098–1101, September 1952.

[12] Terry A. Welch, "A Technique for High Performance Data Compression," *IEEE Computer*, Vol. 17, pp. 8-19, June 1984.