Journal of Education and Science
Vol. 27, No.2, 2018, pp. 48-57
ISSN 1812-125X
http://edusj.mosuljournals.com

Better Adaptive Text Compression Scheme

Duha Amir Sultan
Dept. of Biology/ Education College of Girls/ University of Mosul

تاريخ الاستلام 2012/04/17 تاريخ القبول
2012/11/07

الخلاصة

ظهرت طرقٌ عديدةٌ لكبس البيانات, واحدة من أشـــهر هذه الطرق : هي الطريقة المقترحة من قبل العالمين Ziv و Lempel والتي أطلق عليها اســم LZ77, ثم طورت هذه الطريقة من قبل العالمين Storer , Szymanski لتظهر طريقة جديدة هي LZSS , والتي أعطت نتائج أفضل مقارنة بسابقاتها بعد أن طبقت على نصوص كثيرة وضخمة. في هاتين الطريقتين وجميع الطرق الأخرى التي تنضوي تحت تصنيف الطرق الديناميكية لكبس البيانات يتم مسح الملف من اليسار إلى اليمين لإيجاد أطول تطابق بين مخزن البيانات (النص المشفر مسبقا) والنص الذي سيشفر. الطريقة المقترحة في هذا البحث تعتمد مبدأ البحث بإتجاهين, من اليسار إلى اليمين ثم من اليمين إلى اليســـار, على الرغم أن هذه الطريقة تحتاج وقتاً أطول نوعا ما , إلا أنها أعطت نتيجة كبس اكبر للبيانات.

Abstract

A data compression scheme suggested by Ziv and Lempel, LZ77, is applied to text compression. A slightly modified version suggested by Storer and Szymanski ,LZSS, is found to achieve compression ratios as good as most existing schemes for a wide range of texts. In these two methods and all other dynamic methods, a text file is searched from left to right to find the longest match between the lookahead buffer (the previously encoded text) and the characters to be encoded. The method suggested in this work depends the searching in two directions, from left to right and from right to left, although this process takes more time, better compression results were obtained.

Introduction

Data compression is the business of reducing the amount of space needed to store files on computers or of reducing the amount of time taken to transmit the information over a channel of given bandwidth [7]. Data stored on a computer falls into two groups: First: digital representation of data that is continuous in nature such as images, sounds, video sequences,…., because the form stored is already the quantized version of the original, it is appropriate for further approximation to be permitted and lossy compression techniques can be used to obtain extremely compact representation. Second: data such as text, archival images of historical documents,…. Where the original source of data must be capable of being reconstructed exactly, so lossless compression methods can be used to compact and save these kinds of data [17].

In this work, we concentrate on lossless compression, precisely on adaptive methods (explained in section 3), specially LZSS, which is an improvement of LZ77, a table of previous works (from 1977 to 2005) also presented in this section, a development is applied to LZSS generating a new scheme (named as LZD) with better compression results, the idea of LZD is explained in details in section 4. Finally, and in section 5, some experimental results on some files are showed followed by a simple comparison between LZSS and LZD methods.

Types of Compression Methods

Compression methods can be classified into many groups (Fig. 2.1), as they have been designed for a wide variety of types of information such as text, images, and sound. These usually call for quite different approaches to the problem because of the different types of information they contain.

In general, compression methods are divided into two groups: Lossy and Lossless:

- Lossy, or irreversible, compression is used for digitized analogue signals such as speech and pictures.
- Lossless, reversible or noiseless, compression (where the original can be recovered exactly from it's compressed version) is particularly important for text, since in this situation errors are not exactly accepted [2].

Lossless compression methods, on the other hand, can be assorted to Online and Offline.

- Online methods accomplish the compression in one pass.
- Offline methods process the entire input string several times before the final encoding strategy is determined [25].

Another categorization can be made to static, semiadaptive, and adaptive compression schemes.

- In Static schemes the rules used to encode the string are kept fixed during the process.
- Semiadaptive schemes differ from the previous in using a different model for each encoded text.
- Adaptive (or dynamic) methods changes the rules according to the characteristics of the text, this is normally implemented as an online learning process [2].

Finally, an other assortment can be made, according to the entropy theory, (where the entropy is a measure of the information content of the text to give a limit of the best possible compression) to Entropy and Non-entropy methods.

- Entropy methods are used when the objective is to maximize compression.
- In Non-entropy methods speed of operation, economy of memory usage are desirable features beside the aim of compression [22].

More details can be found in [2], [25], [17], [7], [10], and [22]

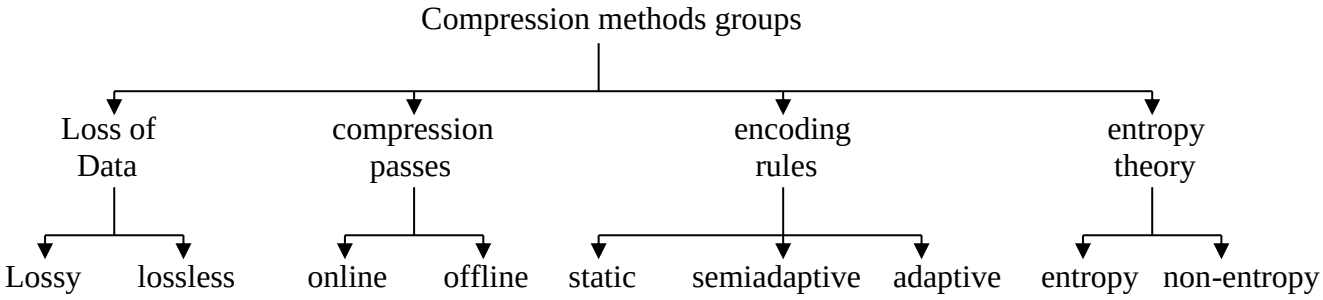

Fig. 2.1 Types of Compression Methods

Adaptive Methods

Almost all practical adaptive encoders are encompressed by a family of algorithms derived from the work of Ziv and Lempel. The essence is that the phrases are replaced with a pointer to where they have occurred earlier in the text. This family of schemes is called Ziv_lempel compression, abbreviated as LZ compression [6]. This method adapt quickly to a new topic, but it is also able to code short function words because they appear so frequently.

Decoding a text that has been compressed in this manner is straight forward; the decoder simply replaces a pointer with the already decoded text that it points to. In practice, LZ coding achieves good compression, and an important feature is that decoding can be very fast.

One form of a pointer is a pair (m,l) that represents the phrase of l characters starting at position m of the input string. The pointer is

constructed from the earlier text of a predefined window. The window may be unrestricted (growing window) or it may restricted to a fixed_size window of the previous N characters, where N is typically several thousands [6].

- The growing window offers better compression by making more substrings available. As the window becomes larger, however, the encoding may slow down because of the time taken to search for matching substrings; compression may get worse because pointer must be larger; and if memory runs out the window may have to be discarded; giving poor compression until it grows again.
- A fixed_size window avoids all these problems, but it has fewer substrings available as targets of pointers. Within the window chosen, limiting the set of substrings that may be the target of pointers makes the pointers smaller and encoding faster.

The table below labels the previous works and the most significant variations of LZ compression, and summarizes the main distinguishing features among them:-

Table 2.1 Principal LZ Variations

LZ77 [27]	(1977)	pointers and characters alternate, pointers indicates a substring in the previous N characters.
LZR [18]	(1981)	pointers and characters alternate, pointers indicates a substring anywhere in the previous characters.
LZSS [2]	(1986)	pointers and characters are distinguished by a flag bit, pointers indicate a substring in the previous N characters.
LZH [3]	(1987)	same as LZSS, except Huffman coding is used for pointers on a second pass.
LZ78 [28]	(1978)	pointers and characters alternate, pointers indicate a previously phrased substrings.
LZW [24]	(1984)	the output contains pointers only, pointers indicates a previously phrased substring, pointers are of fixed size.
LZC [20]	(1985)	the output contains pointers only, pointers indicate a previously phrased substring.
LZT [21]	(1987)	same as LZC, but with phrases in a LRU list.
LZMW [16]	(1984)	same as LZT, but phrases are built by concatenating the previous two phrases.
LZJ [11]	(1985)	the output contains pointers only, pointers indicates a substring anywhere in the previous characters.
LZFG [9]	(1989)	pointers select a node in a trie, strings in a trie are from a sliding window.

LZRW [5]	(1991)	refers to variants of the LZ77 with an emphasis on improving compression speed through the use of hash table.
LZX [5]	(1998)	it was publicly released as an Amiga file archiver.
LZMA [5]	(1998)	uses a dictionary scheme similar to LZ77 with a variable size up to 4GB.
LZWL [5]	(2005)	work with syllables

3.1- LZ77 Scheme:-

LZ77 was the first form of LZ compression to be published [27]. In this scheme, pointers denote phrases in a fixed-size window that precedes the coding position. There is a maximum length for substrings that may be replaced by a pointer , given by the parameter F (typically 10-20). These restrictions allow LZ77 to be implemented using a "sliding window" of N characters. Of these, the first N-F have already been encoded and the last F constitute a lookahead buffer.

To encode a character, the first N-F characters of the window are reached to find the longest match with the lookahead buffer. The match may overlap with the buffer but obviously can not be the buffer itself.

The longest match is then coded into the triple (i,j,a), where i is the offset of the longest match from the lookahead buffer, j is the length of the match, and a is the first character that did not match the substring in the window. The window is then shifted right j+1 characters, ready for another coding step. Attaching the explicit character to each pointer ensures that coding can proceed even if no match is found for the first character of the lookahead buffer [13].

From this notation, the string "a b b a a b b b a b a b"

Lookahead buffer

If the coding process reaches this character

After completing the coding process the output would be:
(0,0,a)(0,0,b)(2,1,a)(1,1,a)(1,3,b)(3,2,b)(8,3,null)

3.2- LZSS Scheme:-

The output of the LZ77 is a series of triples, which can also be viewed as a series of alternating pointers and characters. The use of explicit character followed every pointer is wasteful in practice because it could often be included as part of the next pointer. LZSS addresses this problem by using a free mixture of pointers and characters, the later being included whenever a pointer would take more space than the characters it codes. A window of N characters is used in the same way as for LZ77, so the pointer size is fixed. An extra bit is added to each pointer or character to distinguish between them, and the output is packed to eliminate unused bits [2]. The LZSS algorithm is:

```
While lookahead buffer no empty do
    get a pointer (offset,length) to the longest match
    in the window for the lookahead buffer
if length>p then
    output the pointer (offset,length)
    shift window N characters
else
    output first character in lookahead buffer
    shift window one character
```

where p is the number of characters (or bytes) taken by a pointer [2].
If we take the same string in section 3.1: abbaabbbabab , the coded string would be:
(0,a)(0,b)(0,b)(0,a)(1,1,3),(1,3,2)(1,8,3)

The output pointer contains either two or three elements, the first element in two cases is a single distinguishing bit, if it is 0 means that there is no coding and a complete character would be found in the coded file, if it is 1 a pointer of offset and match length is followed.

4- LZD (the proposed) Scheme:-

LZD is abbreviated from LZSS with two-Dimension search, so it uses the structure of LZSS scheme. LZD encoder is parameterized by N, the size of the window in the text, and F the maximum of the substring that may be replaced by a pointer as in LZSS.

The main difference between the two methods is that in LZSS the searching process for a match is implemented using greedy algorithm and encoding proceeds from left to right, while in LZD the search for a match proceeds from left to right then return back from right to left, this gives better chance to find a longer match between the already encoded string and the previously encoded text. For example, if we take the two words "MACHINE" and "CAMERA", in LZSS there is no similar phrase between them, but if we use LZD, the first underlined phrase of the word CAMERA will simulate the phrase "MAC" of the other word, when backward search is accomplished.

A single bit is added to distinguish whether the substring is coded in forward and backward manner.

More time is needed in using LZD than LZSS scheme, to days this is not very important, as the CPU's have become very cheap and with different high speeds, so all the recent schemes have concentrated on achieving better possible compression rather than the time they take.

The algorithm of LZD is:-

```
While lookahead buffer not empty do
    Get a pointer (offset1,for_length)
```

```
        Get a pointer (offset2,back_length)
    If for_length>back_length then
        Output the pointer(offset1,for_length)
        Shift the window for_length characters
    Else
        Output the pointer(offset2,back_length)
        Shift the window back_length characters
```

If the algorithm is implemented on the string: abbaabbbabab , then the output would be: (0,a) (0,b) (1,0,2,2) (1,1,1,3) (1,1,3,2) (1,1,8,3)

Same as LZSS Pointers contain 4 elements (i,j,k,l)

First bit , i , of the pointer used as LZSS (to distinguish if the output is a character or a pointer), the second bit , j, (which is either 0 or 1) used to distinguish if the match is from left to right (if the bit is 1) or from right to left (if the bit is 0), bits k and l represent the offset and the length of the longest match respectively (as in LZSS).

To decode the compressed string: first the size of the lookahead buffer is zero and a single bit is read from the coded string or file, if it is 0,then the code of a complete character (8 bits) is read (and the size of the lookahead buffer increased by 1), if it is 1, the three element of the pointer j,k,l must be read, and l characters are taken from the lookahead buffer starting at position or character k, depending on the value of j, if it is 0, the search starts from the lookahead buffer down to 0, and vice versa, if j is 1.

Experimental Results

In this section, the result of some experiments with the coding scheme are presented using a variety of different sorts of text files and respectable performance is achieved with all of them. Empirical comparison between the enhanced method and the standard one are also described. These results are obtained by a program written in C++ language.

Below the files used in experiments; the name of each file beside its type is presented:-

1-Huge1, Huge2, Huge3: text files collected from a set of research abstracts. (size: 1Mb-5Mb).

2-Small: a help file taken from C-language package. (17970 characters (≈17KB)).

3-Lzd.c: a commented C program- the same program used in compression, (10649 characters).

4-Data: a collection of characters and numeric data in text format (24000 characters (≈24 KB)).

Table 5.1 shows the tests ran on those files, the size of the file after compression is shown under the file name, the second row of the table shows the compressed file as a percentage of the original using LZSS method. Table 5.2 shows the same tests, but using LZD method.

Table 5.1 LZSS compression performance

File name	Huge1	Huge2	Huge3	small	Lzd.c	data
File size	1.5 MB (1572864) chars.	3MB (3145728) chars.	5MB (5242880) chars.	17970 chars.	10649 chars.	24000 chars.
File size after compression	979894	2105985	4006540	11570	7602	15970
Size after compression (as a percentage)	62.2%	66.9%	76.4%	64.3%	71.3%	66.5%

Table 5.2 LZD compression performance

File name	Huge1	Huge2	Huge3	small	Lzd.c	data
File size	1.5 MB (1572864) chars.	3MB (3145728) chars.	5MB (5242880) chars.	17970 chars.	10649 chars.	24000 chars.
File size after compression	841482	1898029	3544186	10873	6745	12623
Size after compression (as a percentage)	53.4%	60.3%	67.6%	60.5%	63.3%	52.5%

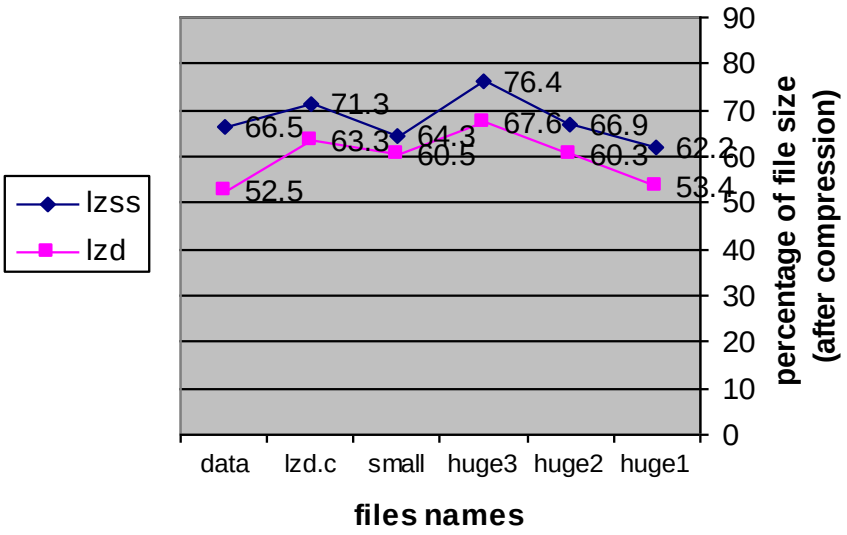

Fig. 5.1 LZSS and LZD compression performance

The two methods are implemented on the same five files with the same window size(N=4096 bytes or characters), which is a middle window size, as if the window becomes larger the search and encoding process may slow down, with smaller window size, the chance of finding a match between the lookahead buffer and the encoded text will be decreased causes the compression performance to be decreased .

From the tables, it has been obviously seen that LZD achieved better compression performance than LZSS, the difference in compression performance between the two methods falls in the range of 4-10%, better result was obtained with the "data" file, means that this file contains more contrast phrases than others.

LZD can be considered as a development of LZSS, which is a type of lossless compression methods, means that there is no loss of data and the file after decompression is completely similar to the original.

Refrences

1- Banikazemi M., "LZB Data Compression with Bounded References", Data Compression Conference (2009).
2- Bell T.C., "Better OPM/L Text Compression", IEEE Transactions on Communication", Vol.COM-34,No.12 (Dec.), 1176-1182 (1986).
3- Brent R.P., "A Linear Algorithm for Data Compression", Australian Computer Journal, Vol.19, No.2, 64-68 (1987).
4- Castelli V., Lastras-Montano L.A., "Bounds on Expansion in LZ77-Like Coding", IEEE Transactions on Information Theory, Vol. 52, Issue 5, 1974-1989 (2006).
5- David S., "data Compression, The Complete Reference", Springer, 4th Edition (2006).

6- Diego Arroyuelo, Gonzalo Navaro, "Smaller and Faster Lempel-Ziv Indices", Citeseer (2010).
7- En-Hui Yang, Kieffer J.C., "Efficient Universal Lossless Data Compression Algorithms Based on a greedy Context-Dependent Sequential Grammar Transform", IEEE Transactions on Information Theory, Vol.46, Issue 3, 755-777 (2000).
8- Ferreira Artur, Oliveira Arlindo, Figueiredo Mario, "Time and Memory Efficient Lempel-Ziv Compression Using Suffix Arrays", arXiv (2009).
9- Fiala E.R., Green D.H., "Data Compression with Finite Windows", Communication of the ACM, Vol.32, No. 4, 940-505 (1989).
10- Istle J., Mandelbaun P., Regentova E., "On line Compression on ASCII Files", Information Technology: Coding and Computing, Vol.1, 755-759 (2004).
11- Jakobsson, M, "Compression of Characters Strings by an Adaptive Dictionary", BIT Vol.32, No.4, 593-603 (1985).
12- Jan Lansky, and Michal Zemlica, "Text Compression : Syllables", 32-45, ISBN 80-01-03204 (2005).
13- Kreft S., Navarro G., "LZ77-Like Compression with Fast Random Access", "Data Compression Conference", 239-248 (2010).
14- Little G., Diamond J., "Optimum String Match Choices in LZSS", Data Compression Conference (2010).
15- Lonardi S., Szpankowski W., Ward M.D., "Error Resilient LZ77 Data Compression: Algorithms, Analysis, and Experiments", IEEE Transactions on Information Theory, Vol. 35, Issue 5, 1799-1813 (2007).
16- Miller V.S., Wegman M.N., "Variations on a Theme by Ziv and Lempel", NATO ASI Series, Vol.F12, Springer-Verlag, 131-140 (1984).
17- Moffat A., Bell T.C., Witten I. H., "Lossless Compression for Text and Images", E.4, Coding and Information Theory, (Oct.), 1-49 (1995).
18- Rodeh M., Pratt V.R., Even S., "Linear Algorithm for Data Compression via String Matching", Journal of the ACM, Vol.28, No.1, 16-24 (1981).
19- Storer J.A., Szymanski T.G., "Data Compression via Textual Substitution", Journal of the ACM, Vol.29, No.4, 928-951 (1982).
20- Thomas S.W., McKie J., Davies S., Turkowski K., Woods J.A., Orost J.W., Compress Program and Documentation, version 4 (1985).
21- Tischer P., "A Modefied Lempel-Ziv-Welch Data Compression Scheme", Australian Computer Science Communication, Vol.9, No.1, 262-272 (1987).
22- Umesh S. Bhadade, Prof. A.I. Trivedi, "Text Compression Methods Based on Dictionaries", International Journal of Computer Applications,Vol.8, Issue:7, 30-37 (2010).
23- Wei-ling Chang, Xiao-Chun Yun, Bin- Xing Fang, Shu-peng Wang, "The Block LZSS Compression Algorithm", Data Compression Conference (2009).
24- Welch T.A., "A Technique for High Performance Data Compression", IEEE Computer, Vol.17, No.6, 8-19 (1984).

25- Witten I.H., Cleary J.G., "Modeling for Text Compression" Bell T M Computing Surveys, Vol.21, No.4, (Dec.), 557-591 (1989).
26- Yuan Jing, "The Combinational Application of LZSS and LZW Algorithms for Compression Based on Huffman", Proceedings of 2011 International Conference on Electronics and Optoelectronics, Vol. 1, Pages:V1-397-V1-399 (2011).
27- Ziv J,. Lempel A., "A Universal Algorithm for Sequential Data Compression", IEEE Transactions on Information Theory, Vol.It-23, No.3, 337-343 (1977).
28- Ziv J., Lempel A., "Compression of Individual Sequences via Variable Rate Coding", IEEE Transactions on Information Theory, It-24, No.5, 530-536 (1978).