Removing Redundancy in Dictionary based Compression Techniques

Neha Gupta[1], Ranjit Kumar[2] and Apoorv Gupta[3]

[1]Gateway Institute of Engineering and Technology, MDU Rohtak, India
[2]Gateway Institute of Engineering and Technology, MDU Rohtak, India
[3]Technological Institute of Technology & Sciences, MDU Rohtak, India
[1]guptaneha2006@gmail.com, [2]ranjitpes@gmail.com, [3]apoorv.gupta@infosys.com

Abstract: Many data compression schemes are developed nowadays and they are selected according to the requirements, such as fast encoding, fast decoding, a good compression performance, small amount of required memory etc. In this thesis, the basic dictionary based data compression techniques i.e. LZ77, LZ78 and LZW, have been studied to find their drawbacks, so that they can be improved further. As out of LZ77, LZ78 and LZW, the variants of LZW are widely used in a number of applications. So, the thesis is mainly oriented towards improving on LZW. Based on the study, we have tried to identify the sources of redundancy in these algorithms and have suggested a method which is a simple dictionary-pruning algorithm that removes the irrelevant entries from the dictionary every time the dictionary is out of space; to store new phrases. This ensures that the dictionary is always adaptive.

Keyword: Compression, Decompression, Dictionary-based, LZW, Pruning, Performance.

1. Introduction

Data compression is, in the context of computer science, the science (and art) of representing information in a compact form. It has been one of the critical enabling technologies for the ongoing digital multimedia revolution for decades. Most people frequently use data compression software such as zip, gzip and WinZip (and many others) to reduce the file size before storing or transferring it in media. There are two major families of compression techniques when considering the possibility of reconstructing exactly the original source. They are called *lossless* and *lossy* compression. A compression approach is lossless only if it is possible to exactly reconstruct the original data from the compressed version. A compression method is lossy if it is not possible to reconstruct the original exactly from the compressed version. Lossless data compression is generally implemented using one of two different types of modeling: statistical or dictionary-based. Statistical modeling reads in and encodes a single symbol at a time using the probability of that character's appearance. Dictionary-based modeling uses a single code to replace strings of symbols. In dictionary-based modeling, the coding problem is reduced in significance, leaving the model supremely important.

2. Dictionary-based modeling (LZW)

The LZW method starts by initializing the dictionary to all the symbols in the alphabet. Then the encoder inputs symbols one by one and accumulates them in a string '*word*'. After each symbol is input and is concatenated to '*word*', the dictionary is searched for string '*word*'. As long as '*word*' is found in the dictionary, the process continues. At a certain point, adding the next symbol '*x*' causes the search to fail; string '*word*' is in the dictionary but string '*word*' + '*x*' (symbol '*x*' concatenated to '*word*') is not. At this point the encoder outputs the dictionary pointer that points to string '*word*', Saves string '*word*' + '*x*' (which is now called a *phrase*) in the next available dictionary entry, and Initializes string '*word*' to symbol '*x*'.

Since the first 256 entries of the dictionary are occupied right from the start, pointers to the dictionary have to be longer than 8 bits. A simple implementation would typically use 16-bit pointers, which allow for a 64K-entry dictionary (where 64K = 216 = 65,536). Such a dictionary will, of course, fill up very quickly in all but the smallest compression jobs. Another interesting fact about LZW is that strings in the dictionary get only one character longer at a time. It therefore takes a long time to get long strings in the dictionary, and thus a chance to achieve really good compression. We can say that LZW adapts slowly to its input data.

The encoding algorithm is:

```
word ← ''
while not EOF do
   x ← read_next_character()
   if word + x is in the dictionary then
      word ← word + x
   else
      output the dictionary index for word
      add word + x to the dictionary
      word ← x
   end if
end while
output the dictionary index for word
```

The decoding algorithm now is:

read a codeword *x* from the compressed file

look up dictionary for phrase at *x*
output phrase
word ← *phrase*
while not EOF do
  read *x*
  look up dictionary for phrase at *x*
  if there is no entry yet for index *x* then
    phrase ← *word* + *firstCharOfword*
  end if
  output *phrase*
  add *word* + *firstCharOfphrase* to the dictionary
  word ← *phrase*
end while

3. The Dictionary Pruning Algorithm-LWZ(P)-Proposed Work

In this section a method for dictionary pruning has been proposed. As LZW is a very popular dictionary based data compression technique, modification attributes to include our pruning process.

In LZW, phrases from input string are added to dictionary and corresponding 12 bit codes are sent to the output. So, an LZW dictionary can contain maximum of 2^{12} = 4096 entries. The basic LZW algorithm is modified in such a way that whenever the dictionary gets full, a function is called that will remove all the entries that have never been used till time, since the creation of dictionary. The main work of the function is to identify these phrases. For this, every entry in dictionary is associated with a flag value. The function checks every phrase for its flag value, and removes it if the flag value matches the deletion condition. Values of flag variable according to specific condition are:

dict[i].flag	=	0	unused entries
		1	entry used at least once
		2	deleted entry

3.1 Assumptions

Table 1. Assumptions table for dictionary pruning algorithm.

Symbol	Meaning
word	string that contains all the characters that have been scanned till time and should be searched in the dictionary
x	next character to be scanned from the input file
size	number of phrases that are currently present in the dictionary

3.2 Algorithm

The pruning process algorithm work as follows:

a) Scan the input string, character by character until the end of file is reached.
b) After each character '*x*' is input, it is concatenated to '*word*', and the dictionary is searched for string '*word*'.
c) As long as '*word*' is found in the dictionary, the search process continues.
d) At a certain point, adding the next symbol '*x*' causes the search to fail; string '*word*' is in the dictionary but string '*word*' + '*x*' (symbol '*x*' concatenated to '*word*') is not.
e) At this point the encoder outputs the dictionary pointer that points to string '*word*', and saves string '*word*' + '*x*' (which is now called a *phrase*) in the next available dictionary entry, and initializes string '*word*' to symbol '*x*'.
f) This process continues until the dictionary is full i.e. all 4096 locations have been occupied by phrases.

As the dictionary overflows, all the entries having flag = 0 i.e. the phrases whose code has never been used in the output, are searched and removed from the dictionary by setting the corresponding flag value to 2.
Now to insert new entries in the dictionary, the space restored during deletion, is used.

3.3 Pseudo-Code

```
LZW(P)
word ← ''
while not EOF do
  x ← read_next_character()
  if word + x is in the dictionary then
    word ← word + x
  else
    search for the first occurring available location
    add word + x to the dictionary
    size ← size + 1
    output the dictionary index for word
    word ← x
  end if
  if size = 4096
    dict_prune()
 end while
 output the dictionary index for word
dict_prune()
for all dictionary entries do
  if flag is 0
    set flag ← 2     //marks the entry as deleted
    size ← size – 1
```

3.4. Advantages

The dictionary pruning algorithm proposed above has the following advantages:

a) In classic version of LZW, the dictionary becomes static when it reaches its maximum size, but the proposed algorithm remains adaptive, as whenever the dictionary reaches its maximum value, it removes the unused phrases from the dictionary.

b) The proposed algorithm improves the compression by a considerable amount, as it is ensures that dictionary contains only those phrases that will help in compression.
c) The irrelevant entries are always updated, making space for new entries that are more relevant to the input file.

4. Performance Evaluation

4.1 Comparison of LZW and LZW(P)

This section presents compression and analysis results of a classic and proposed LZW algorithm on a number of different files. Table 2 gives the details of the Datasets on which the above algorithms have been tested.

Table 2. Datasets

S. No.	File Name	File Size(bytes)
1	Book1.txt	768770
2	Book2.txt	610855
3	News.txt	377108
4	Paper.txt	53155

Figure 1 shows the overall compression achieved as the source files were processed using the two algorithms. The two different bars correspond to classic LZW and LZW(P) algorithms. The bars shows that compression using LZW(P) has a consistent advantage over LZW:

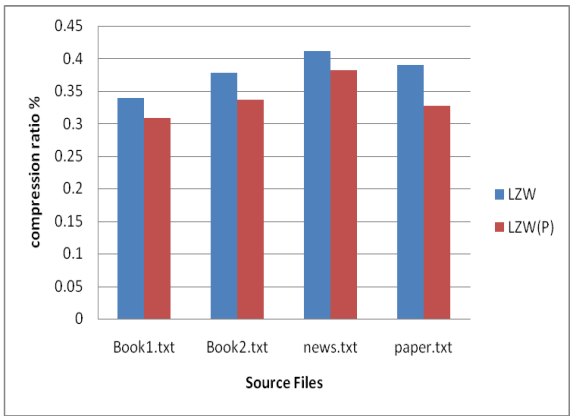

Figure 1. LZW (P) performance graph compared to LZW compression.

The graph shows the compression gain that LZW(P) has over LZW. In real scenario, the LZW(P) shows a gain of about 6-8% in compression ratio when tested on different files.
Some experimental results to show the effect of dictionary pruning on compression performance are given in Table 3.

Table 3. Performance Analysis of LZW and LZW(P).

Source Files	Original Size (Bytes)	LZW (Bytes)	LZW(P) (Bytes)
Book1.txt	768770	260536	237580
Book2.txt	610855	231020	205560
News.txt	377108	155260	143902
Paper.txt	53155	20696	17398

5. Conclusion

The various data compression techniques and methods to optimize them were considered. The first algorithm that is proposed takes into account the fact that the dictionary used in LZW becomes static once all the 4096 locations has been occupied. The algorithm adds a process for dictionary pruning to LZW, so that it remains adaptive. It does so by removing the entries that are irrelevant and are not required. These entries take up unnecessary dictionary space that could be utilized by more useful keywords. The proposed algorithm removes these entries whenever the dictionary is full. Waste phrases are found by associating each phrase with a flag value which is 0 for the phrases that were never used during compression. By applying this modification better compression ratios were achieved. So, by adding a little extra overhead, the proposed method achieved about 6%-8% better compression ratios than the classic LZW.

References

[1] C. L. Yu and J. L. Wu, "Hierarchical dictionary model and dictionary management policies for data compression", Signal Processing, Sept 1999.

[2] R. N. Horspool, "The Effect of Non-Greedy Parsing in Ziv-Lempel Compression Methods", IEEE Data Compression Conference, 1995.

[3] S. Subathra, M. Sethuraman and J. V. B. James, "Performance Analysis of Dictionary based Data Compression Algorithms for High Speed Networks", IEEE Indicon Conference, Dec 2005.

[4] N. Zhang, T. Tao, R. V. Satya and A. Mukherjee, "Modified LZW Algorithm for Efficient Compressed Text Retrieval", draft, Computer Science Dept., Univ. of Central Florida, 2004.

International Journal of Computer Science & Emerging Technologies (E-ISSN: 2044-6004)
Volume 1, Issue 4, December 2010

[5] R. N. Horspool, "Improving LZW", Research, Dept. of Computer Science, University of Victoria, Victoria, B.C, Canada.

[6] Y. Matias, N. M. Rajboot and S. C. Sahinalp, "The Effect of Flexible Parsing for Dynamic Dictionary Based Data Compression", Proceedings of the Data Compression Conference, 1999.

[7] D. A. Huffman, "A method for the construction of minimum redundancy codes", Proceedings IRE 40, Sept 1952.

[8] D. Salomon, "Data Compression - The Complete Reference", 2nd Ed, Springer-Verlag New York, Inc., New York, 2001.

Author Biographies

Neha Gupta Her Birth place is Punjab and Date of Birth is 2 November 1983.She is B.tech in Information Technology from TITS,Bhiwani (2005 batch) and M.Tech in Computer Science from BITS, Bhiwani (2010 batch). Now she is working as an Asst. Professor in GIET, Sonipat having 3.5 years teaching experience.

Ranjit Kumar His Birth place is Bihar and Date of Birth is 20 November 1981.He is B.tech in Computer Science from P.E.S, Maharashtra (2005 batch) and M.E. in Software Engineering from BIT Mesra, Ranchi (2008 batch). Now he is working as an Asst. Professor in GIET, Sonipat having 2.5 years teaching experience.

Apoorv Gupta His Birth place is Haryana and Date of Birth is 7 October 1987.He is B.tech in Computer Science from TITS, Bhiwani (2009 batch). Now he is working in Infosys Technology, pune.