World Academy of Science, Engineering and Technology
International Journal of Computer and Information Engineering
Vol:3, No:11, 2009

Study of Efficiency and Capability LZW++ Technique in Data Compression

Yusof. Mohd Kamir, Mat Deris. Mohd Sufian, and Abidin. Ahmad Faisal Amri

Open Science Index, Computer and Information Engineering Vol:3, No:11, 2009 publications.waset.org/1431/pdf

***Abstract*—**The purpose of this paper is to show efficiency and capability LZW++ in data compression. The LZW++ technique is enhancement from existing LZW technique. The modification the existing LZW is needed to produce LZW++ technique. LZW read one by one character at one time. Differ with LZW++ technique, where the LZW++ read three characters at one time. This paper focuses on data compression and tested efficiency and capability LZW++ by different data format such as doc type, pdf type and text type. Several experiments have been done by different types of data format. The results shows LZW++ technique is better compared to existing LZW technique in term of file size.

***Keywords*—**Data Compression, Huffman Encoding, LZW, LZW++, RLL, Size.

I. Introduction

Nowadays, data is important for business organization. Some organization is needed a good software to compress the data. Research about existing algorithm is needed to produce an enhanced algorithm or new algorithm for data compression. The software for data compression is useful because it can help to reduce hard disk space or transmission bandwidth. This software can used to reduce size of data. Suppose original data is 100Mb, after compression process, a new data size is 70Mb. In this case, this software can reduce 30% data size from the original data size.

Data compression also is important for data storage and data transmissions. The purpose of compression is to reduce original data size. Smaller data size is very efficient for data transmission from one location to another location. Time for transmission also is faster and effective. That way, smart software for data compression is needed for business organization to ensure business operation can performed efficiently and effectively.

In this paper, three existing techniques or algorithms for data compression have been studied. The purpose of this study is to find a good algorithm for data compression. After that, enhancement is made to produce enhanced or new algorithm for data compression.

II. Previous Method

Huffman encoding is one of the data compression techniques [1]. The first steps in this technique is read file. Second steps, characters in data file are converted to a binary code. Third steps, this technique will assign of keys to source message based on probabilities in the message. This technique has been applied for data compression. It is efficient for image compression such bmp format, tiff format and gif format. The results shows new data size is smaller compared to original data size. However, this technique is not suitable for text compression because it is not stable. Sometimes, the results shows new size is bigger compared to original size.

Another technique in data compression is Run-Length-Encoding (RLE) [2][3]. This technique is sometimes knows as "*run-length-encoding*". First steps in this technique is read file. After that, it will scan through the file and find the repeating string of characters. After repeating characters is found, it will store using escape character (ASCII 27) followed by the character and a binary count of the number of items it is repeated. Smart compression software is needed to compress strings of two or more repeated characters. The new size is bigger compared to original size if software is not smart. This technique is useful for image especially for solid black picture bits. This technique also is efficient for repeating of characters. However, this technique is not efficient and effective if data file has less repeating of characters. Sometimes, the results shows new data size is bigger compared to original data size.

Third technique have been studied in data compression are Lempel Ziv Welch (LZW) [4]. This technique has been applied for data compression. A first step in this technique is read data file. After that, it will read one by one character in file. Then, each character will assign a code. If it is found the same characters, it not assigned a new code but use the existing code in data dictionary. LZW algorithm will loop until characters in data file are null. The LZW algorithm is efficient for data compression such doc type, pdf type and txt type. The results also shows LZW technique can reduce original size to smaller compared to original size.

After studied three techniques for data compression, LZW technique is better compared to Huffman Encoding and Run-Length-Encoding in term data size. However, the LZW has a potential to improve with modification of the algorithm. The modification of LZW algorithm is needed to produce a better result in term of data size.

III. The Proposed LZW++ Method

In this paper we shall propose a new method for data

compression. LZW method is used for platform to produce LZW++ method. Figure 1 and figure 2 shows proposed compression and decompression LZW++ algorithm.

A. Compression Algorithm

Figure 1 shows proposed LZW++ algorithm for compress data.

```
filename = frmMain.dlgOpenFile.FileTitle
filelen = Len(filename)
fileformat = Right(filename, 4)
tfilename = cdlg.FileTitle
tfilelen = Len(tfilename)
tfileformat = Right(tfilename, 4)

If txtOut.Text = "" Then
MsgBox "Select location and file to be
saved.", vbInformation, "Save file is
missing."
txtOut.SetFocus
ElseIf txtOut.Text <> "" Then
If FileExists(txtOut.Text) Then
If MsgBox("File with this name is already in
use. Overwrite existing file?", vbYesNo,
"Overwrite existing file?") = vbYes Then
Kill txtOut.Text
Else
Exit Sub
End If
End If

If fileformat = ".lzw" Then
MsgBox "File is already compressed.",
vbInformation, "File compressed."
Exit Sub
End If


Dim i, sz, nf As Long
nf = List1.ListCount - 1
ReDim Files(nf) As String

For i = 0 To nf
Files(i) = List1.List(i)
Next i
lblInfo.Caption = "Status :" + vbCrLf +
"Compressing " & (nf + 1) & " file(s)..."
TimeStart = GetTickCount
sz = PackageFile(Files, txtOut.Text)
TimeEnd = GetTickCount
time = TimeEnd - TimeStart
lblInfo.Caption = "Status :" + vbCrLf +
"File(s) compressed successfully. File Size:
" & sz & " bytes." + vbCrLf + "Compression
took " & time & "ms."
pbStatus.Value = 0
```

Fig. 1 Propose LZW++ Algorithm for Compress Data

Figure 1 show how LZW++ worked. The LZW++ technique read the data file and compress data into LZW++ format. First, the LZW++ algorithm read the data file. Then, read three characters at one time. Differ with existing LZW technique, where the LZW read one by one character at one time. After that, it will assign code for each three characters and store into data dictionary. Finally, a new data size will calculated to ensure new size is smaller compared to original size.

Example 1:

1. Suppose data file contains character: *SASBDSAQP*
2. Read three characters at one time.
3. Loop until characters in data file are null.

TABLE 1 Data Dictionary

Characters	Code
SAS	1
BDS	2
AQP	3

Original size: X = number of characters
Suppose size for one character is Y = 2kb.

$$\alpha = x * y \quad (1)$$

New size: X = number of characters
Suppose size for one characters is Y = 2kb

$$\beta = x * y \quad (2)$$

There result shows,

$$\beta < \alpha \quad (3)$$

As conclusion, LZW++ technique can reduce size of data. Based on formula 3, new size is smaller compared to original size.

Open Science Index, Computer and Information Engineering Vol:3, No:11, 2009 publications.waset.org/1431/pdf

Open Science Index, Computer and Information Engineering Vol:3, No:11, 2009 publications.waset.org/1431/pdf

B. *Decompress Algorithm*

Figure 2 shows proposed LZW++ algorithm for decompress data.

```
Dim c As Long
Dim TimeStart, TimeEnd, time, sz As Long
Dim filename As String
Dim filelen As Integer
Dim fileformat As String

filename = cDlg.FileTitle
'filename = frmMain.dlgOpenFile.filetitle
filelen = Len(filename)
fileformat = Right(filename, 4)
If txtLZW.Text = "" Then
MsgBox "There is no file to be extracted.",
vbInformation, "File is required."
txtLZW.SetFocus
ElseIf txtLZW.Text <> "" Then
If fileformat <> ".lzw" Then
MsgBox "Invalid type of file" + vbCrLf +
"File " & filename & " cannot be extracted.",
vbInformation
ElseIf fileformat = ".lzw" And txtPath.Text
<> "" Then
lblInfo.Caption = "Status :" + vbCrLf +
"Extracting files..."
TimeStart = GetTickCount
c = ExtractFiles(txtLZW.Text, txtPath.Text)
TimeEnd = GetTickCount
time = TimeEnd - TimeStart
If c <> -1 Then
lblInfo.Caption = "Status :" + vbCrLf + "" &
c & " files extracted." + vbCrLf +
"Extracting of file took " & time & "ms."
Else
lblInfo.Caption = "Status :" + vbCrLf +
"Invalid file format or an internal error has
occured."
End If
ElseIf txtPath.Text = "" Then
MsgBox "Location where the file to be
extracted is required.", vbInformation,
"Location is required."
txtPath.SetFocus
End If
pbStatus.Value = 0
End If
```

Fig. 2 Proposed LZW++ Algorithm for Decompress Data

Figure 2 shows the proposed LZW++ algorithm for decompress data. This algorithm will decompress new size to original size by referring data dictionary.

Example 2:

TABLE 2 Data Dictionary

Characters	Code
SAS	1
BDS	2
AQP	3

Table 2 shows data dictionary after compression process. This proposed algorithm will refer data dictionary to convert new size of data to original size of data.

New size of Data = 123
= 6kb

Original Size = SASBDSAQP
= 18kb

IV. Performance Results LZW++ Method

Performance of LZW++ method is measured in term file size. Table 2 shows input parameters are used for experiments.

TABLE 2 Input Parameters

Data Type	Data Format
Text	doc format, pdf format and txt format.

The same data format and data size are used for experiments. This is important must be considered to ensure accuracy and correctness of the results.

Experiments are tested LZW++ method using doc format, pdf format and txt format. The comparison between LZW and LZW++ is measured in term of file size.

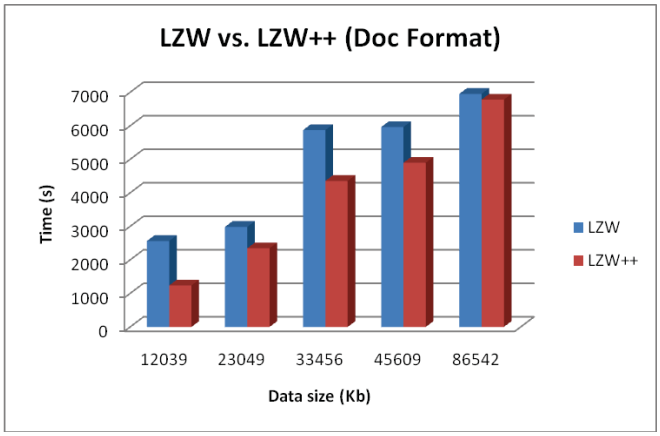


Fig. 3 LZW vs. LZW++ (Doc Format)

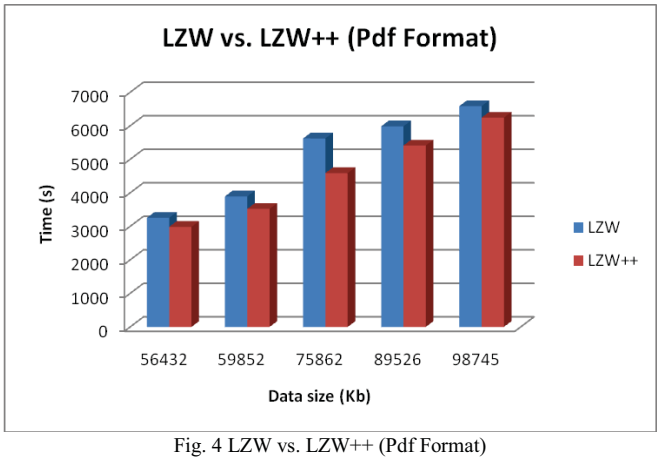


Fig. 4 LZW vs. LZW++ (Pdf Format)

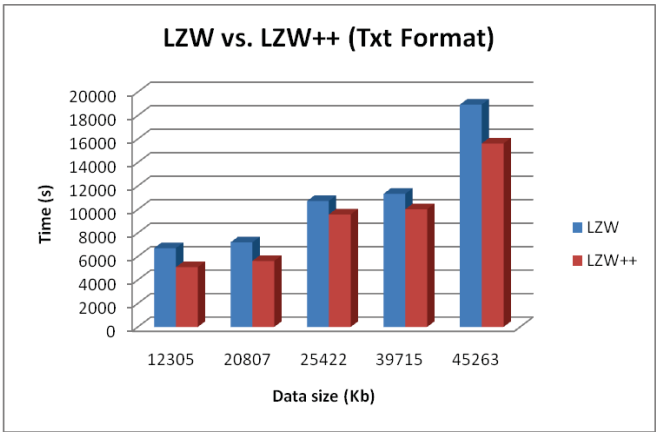


Fig. 5 LZW vs. LZW++ (Txt Format)

Based on figure 3, figure 4 and figure 5 the results shows LZW++ technique is better compared to LZW technique. The efficiency and capability of LZW++ technique is proved by three experiments. A new file size is smaller compared to original size after compression process.

The LZW++ is efficient and effective for data compression especially for doc type, pdf type and txt type.

V. Conclusion and Future Work

This work demonstrates a methodology for compress and decompresses data using LZW++ technique. The modification of LZW algorithm is needed to produce LZW++ technique. The purpose of the modification is to ensure new algorithm or LZW++ able to produce better results compared to existing LZW algorithm. Several experiments are needed to execute in measuring performance and capability of LZW++ technique.

As a conclusion, the results shows LZW++ technique is efficient for data compression. This technique has a potential to use in real environment especially in business organization. Besides that, LZW++ also has a potential to implement for image compression.

Acknowledgment

Thanks in advance for the entire worker in this project, and the people who support in any way, also I want thank Universiti Darul Iman Malaysia (UDM) for the support they offered.

References

[1] Snowbirh, U. (2000). Data Compression Conference (DD'00.
[2] Mohammad Al-Laham and Ibrahim M.M. EL Emary. (2007). Comparative Study between Various Algorithm of Data Compression Technique. 284-290.
[3] Gilbert, H. (1996). Data Image Compression Tools and Technique. WILEY : 68-343.
[4] Grawthrop, J. and W. Liuping. (2005). Data Compression for Estimation of The Stable and Unstable Linear Systems. Automica, 41: 1313-1321.
[5] Blelloch, E. (2002). Introduction to data Compression. Computer Science Department, Garnegie Mellon University.
[6] Grawthrop, J. and Luiping W. (2005). Data Compression for estimation of the physical parameters of stable and unstable linear system. Automatice, 41: 1313 – 1321.
[7] Ian, H.H. Witten, Moffat, a. and Bell, T.C. (1999). Managing Gigabytes: Compressing and Indexing Documents and images.
[8] Bentley, J.L., Sleator, D.D., Tarjan, R.E., and Wei, V.K. (1986). A Locally Adaptive Data Compression Scheme. Commun. ACM 29, 4 (Apr.), 320-330.
[9] Ngoc, V. and Alistair, M. (2006). Improved wordaligned binary compression for text indexing. IEEE Trans. Knowledge & Data Engineering, 18: 857-861.



Mohd Kamir Yusof obtained her Master of Computer Science from Faculty of Computer Science and Information System, Universiti Teknologi Malaysia in 2008. Currently, he is a Lecturer at Department of Computer Science, Faculty of Infomatics, Universiti Darul Iman Malaysia (UDM), Terengganu, Malaysia.



Mohd Sufian Mat Deris obtained her Master of Education (educational technology) from Faculty of Education, Universiti Teknologi Malaysia in 2006. Currently, he is a Lecturer at Department of Multimedia, Faculty of Infomatics, Universiti Darul Iman Malaysia (UDM), Terengganu, Malaysia.



Ahmad Faisal Amri Abidin obtained her Master of Computer Science from Faculty of Computer Science and Information Technology, Universiti Putra Malaysia in 2008. Currently, he is a Lecturer at Department of Computer Science, Faculty of Infomatics, Universiti Darul Iman Malaysia (UDM), Terengganu, Malaysia.

Open Science Index, Computer and Information Engineering Vol:3, No:11, 2009 publications.waset.org/1431/pdf