A Study of on Extension Compression Algorithm of Mixed Text by Hangeul-Alphabet

Kang-yoo Ji[1] Mi-nam Cho[2] Sung-soo Hong[3] and Soo-bong Park[4]

[1,2,4] Department of Information & Communication Engineering,
Dongshin University #252 Daeho-dong, Naju, Jeonnam
520-714 Republic of Korea
Tel.:+82-613-330-3193, Fax.: +82-613-330-2909
e-mail : neobacje@bic.re.kr
[3] Department of Computer Information Science DongKang College
#771 Duam-dong Bukgu, Gwang Ju 500-714 Republic of Korea
Tel. +82-62-520-2263, Fax.:+82-62-520-2216

Abstract : This paper represents a improved data compression algorithm of mixed text file by 2 byte completion Hangeul and 1 byte alphabet form.

Original LZW algorithm efficiently compress a alphabet text file but inefficiently compress a 2 byte completion Hangeul text file.

To solve this problem, data compression algorithm using 2 byte prefix field and 2 byte suffix field for compression table have developed. But it have a another problem that is compression ratio of alphabet text file decreased.

In this paper, we proposes improved LZW algorithm, that is, compression table in the Extended LZW(ELZW) algorithm uses 2 byte prefix field for pointer of a table and 1 byte suffix field for repeat counter. where, a prefix field uses a pointer(index) of compression table and a suffix field uses a counter of overlapping or recursion text data in compression table.

To increase compression ratio, after construction of compression table, table data are properly packed as different bit string in accordance with a alphabet, Hangeul, and pointer respectively. Therefore, proposed ELZW algorithm is superior to 1 byte LZW algorithm as 7.0125 percent and superior to 2 byte LZW algorithm as 11.725 percent.

This paper represents a improved data Compre -ssion algorithm of mixed text file by 2 byte completion Hangeul and 1 byte alphabet form.

This document is an example of what your camera-ready manuscript to ITC-CSCC 2002 should look like. Authors are asked to conform to the directions reported in this document.

1. Introduction

To express data size in small space and increase transmission speed in data transmission and store, studies on data compression have been actively developed.

Data compression programs developed include ICE, AXE, PKZIP, LHA(Lempel Zev Huffman), and LZW(Lempel Ziv Welch). Such software has been developed as fitted program to compress text file or execution file with alphabet ASCII code or EBCDIC code system, processed by 1 byte. Thus, to compress 2-byte WanSeongHyeong Hangeul or Hangeul-alphabet mixed text file, compression rate is quite decreased.

Although much has been studied on compression algorithm needed to Hangeul compression, their focuses are on extending LZW compression algorithm with two directions. First, 2-byte WanSeonghyeong Hangeul is separated into 1-byte unit to compress like alphabet. Secondly, compression is executed by 2-byte unit, the size of WanSeongHyeong Hangeul. In the former, as Hangeul-alphabet mixed file is separated into 1-byte unit, it is not efficient to compress 2-byte WanSeongHyeong Hangeul. The latter considers Hangeul and alphabet as 2 byte, so it is very effective to compress Hangeul, while it is not very good for alphabet compression.

Therefore, when 2-byte WanSeongHyeong Hangeul and 1-byte alphabet are compressed by 1-byte unit, this study analyzed existing LZW compression algorithm (LZW1) and LZW2 algorithm, which can effectively compress them, and suggested new algorithm to fit with compression of Hangeul-alphabet mixed text.

This document is a version of the instructions for preparing copies for the final ITC-CSCC 2002 proceedings. The format here described allows for a graceful transition to the style required for that publication.

This document is an example of what your camera-ready manuscript to ITC-CSCC 2002 should look like. Authors are asked to conform to the directions reported in this document.

2. Analysis of LZW WanSeongHyeong Hangeul compression algorithm

2.1 WanSeongHyeong Hangeul compression algorithm (Method 1)

LZW compression algorithm to compress Hangeul text file uses greedy parsing algorithm to divide into string(string) and makes input data binary-coded. Every element of string table within storage location to execute compression consists of 2-byte high-order pointer(p) and 1-byte low-order character(c). This is the method which uses initial LZW algorithm without transformation to compress Hangeul text file.

<Fig. 1> Compression table of LZW compression algorithm

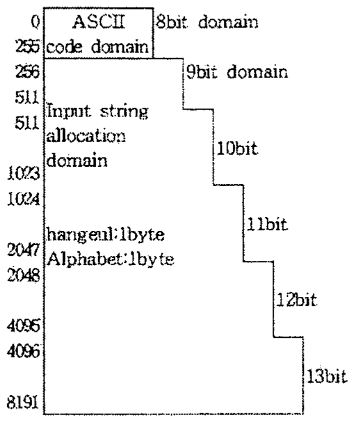


Table No.	ASCII Character
. . .	. . .
32	blank
65	A
66	B
67	C
97	a
177	▒
199	╟
209	╤
219	█
. . .	. . .

CODE	String	
	prefix (2byte)	suffix (1byte)
257	65(A)	(66)B
258	66(B)	67(C)
259	67(C)	32( )
260	32( )	199(╟)
261	199(╟)	177(▒)
262	177(▒)	209(╤)
263	209(╤)	219(█)
264	260	177(▒)
265	263	97(a)
266	. . .	. . .

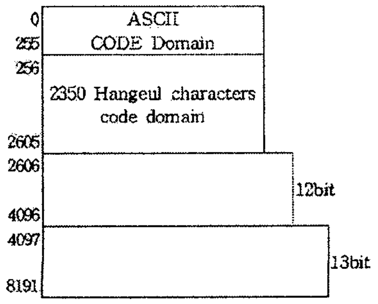
<Table 1> Initial compression table
<Table 2> ASCII Character

As shown in Table 2, '한' and '글' which are 2-byte WanSeongHyeong Hangeul are not processed by 2-byte unit and divided into high-order and low-order byte to correspond with extended ASCII code value. As 2-byte WanSeongHyeong Hangeul is divided to be registered in compression table, it cannot recognize Hangeul. When processing the same input data size, it occupies more compression table than alphabet and it results in low compression rate.

2.2 WanSeongHyeong Hangeul compression algorithm(Method 2)

To effectively compress 2-byte WanSeongHyeong Hangeul, the code of 2-byte WanSeongHyeong Hangeul is initialized at the high-order domain of initial compression table.

<fig 2.> compression table of WanseongHyeong 2-byte Hangeul



As shown in Fig. 2, this method forms compression table by 2-byte unit and can improve compression efficiency in constructing WanSeongHyeong 2-byte Hangeul in the compression table. But as 2350 WanSeongHyeong Hangeul characters have to be constructed in initial compression table, a dictionary for actual compression starts from address 2606. For example, supposing initial compression table for the string of "ABC 한글 한글 a 한글과" is formed as Table 3, the string "ABC 한글 한글 a 한글과" is stored in 2-byte compression table as Table 4.

Table No. (2byte)	ASCII/Hangeul (2byte)
. . .	. . .
32	blank
65	A
66	B
67	C
97	a
. . .	. . .
1280	과
1300	글
2209	한

Code	String	
	prefix (2byte)	suffix (2byte)
2606	A	B
2607	B	C
2608	C	<32>
2609	<32>	한
2610	한	글
2611	글	<32>
2612	2609[<32>한]	글
2613	글	a
2614	a	한
2615	2610[한글]	과
2616	. . .	. . .

<Table 3> Initial compression table
<Table 4> Compression table

As shown in Table 4, compression efficiency is considerably increased in Hangeul compression. In alphabet, 1 byte is enough allocation for suffix for compression table, but 2 byte has to be used to be allocated due to Hangul initial table, so compression table size gets bigger relatively, resulting in increase in compression file size.[2]

On the other hand, this algorithm has to allocate the domain for Hangeul code in compression table, dictionary construction starts from address 2606. Thus, 12~13 bit domain is used for compression space in compression table and compared to Method 1 using 9~13 bit domain, it has the disadvantage that the length of compression file is increased because the length of compression string is increased. In conclusion, when Method 1 and Method 2 are compared in compression efficiency, trade off relation is established in terms of the unit for processing Hangeul and string length. In particular, it is necessary to decrease the efficiency in compressing Hangeul/alphabet mixed text string.

Accordingly, to effectively compress both WanSeongHyeong Hangeul and alphabet, this study took the advantages of Method 1 and Method 2. Prefix of compression table allocates 2 byte to the pointer for Hangeul and alphabet and suffix allocates 1 byte to decrease the size of each element of the compression table. Suffix is not used for the space allocating character, but for the counter(계수) part calculating the number of character which corresponds continuously, when the character read from input file is registered in prefix in order to decrease the size of compression file.

3. Design of compression algorithm of suggested WanSeongHyeong Hangeul/alphabet mixed text document

3.1 Formation of compression table

To compress WanSeongHyeong 2-byte Hangeul and ASCII character effectively, as the above Method 1, suggested ELZW algorithm uses 2-byte pointer for high-order 2 byte of prefix of compression table to express each address of ASCII character and WanSeongHyeong 2-byte Hangeul. Further low-oder 1-byte suffix of compression table is used as counter to calculate the number of character which continuously corresponds with the character registered in read string and the internal of compression table, when the character read from input file is already registered in the compression table.

It is used as the domain to compress input file the element from address 2606 to 8191, after alphabet and WanSeongHyeong Hangeul are initialized in the high-order domain of compression table. If either alphabet or Hangeul is already registered in the compression table, the same character is read from input file, and only one character appears, not repetitive character which corresponds with the address of compression table, the address of initialized character is registered in prefix and not the address of compression table after address 2606 which has the address of relevant character and suffix is initialized as 0. When this method is used, smaller-sized address than the variable address of compression table can be outputted and compression efficiency can get increased when compression file is created from the data stored in the compression table. The algorithm is as follows to register input character in the compression table.

[The algorithm to register input character in the compression table]

```
while((ch = getc(input_file)) != EOF)
{
search ch in TABLE;
if(ch is not exist in current_index of TABLE) {
  Set address of ch in prefix of TABLE[current_index];
  Set 0 in suffix of TABLE[current_index];
}
else {
  move position to existed address of ch in TABLE;
  while((ch = getc(input_file)) != EOF) {
    if( ch is equal to TABLE[position] ) {
     increment position;
     increment count;
    } else {
    if(count > 1) {
     Set address of ch in prefix of TABLE[current_index];
     Set count in suffix of TABLE;
    } else {
     Set address of ch in prefix of TABLE[current_index];
     Set 0 in suffix of TABLE;
increment current_index;
```

Code	String	
	prefix (2byte)	suffix (1byte)
2606	65[A]	0
2607	67[B]	0
2608	68[C]	0
2609	32	0
2610	C3D1[한]	0
2611	B1DB[글]	0
2612	2609	3
2613	97[a]	0
2614	2610	2
2615	B0FA[과]	0
2616	32	0
2617	2606	8
2618	. . .	. . .

<Table 5> Compression Table

That is, the algorithm which registers string in the compression table of Table 5 first reads character sequently from input file to the end of the file to be registered in the compression table. If the read character is not registered in the compression file, it registers the address of relevant character in prefix of compression table and sets suffix of counter field as 0.

But, if relevant character is already registered in certain location within the input table, it repeats the process of comparing the address of character in next location of compression table by reading next character from input file until they do not correspond. If they do not correspond, the address which first corresponds and corresponding counter will be stored in prefix and suffix of table.

To obtain more efficient compression rate, it is good to remove and pack unnecessary bit which exists in the compression table. For example, in Method 2, if alphabet is stored in suffix, only low-order byte may be used. But when compression file is created, both low-order and high-order byte are outputted and compressed file contains unnecessary 1 byte. It is because the data registered in the compression table is not packed at all and all of them are outputted. Thus, when the data with unnecessary bit is outputted as compression file in compression table, it is desirable to remove the unnecessary bit.

[Algorithm to remove unnecessary bit from the compression table]

```
index = 2606;

. . .
packing( )
{
while( TABLE_SIZE > index && index <= LIMIT) {
 read an element from TABLE[index];
 switch( prefix ) {
    case 'table_address'  : write prefix to output_file;
                    write suffix to output_file;
    case 'english_address' :
                  write high byte of prefix to output_file;
    case 'hangeul_address': write prefix to out_file;
      }
    }
}
```

3.2 Decryption of compression file

The decryption of compression file uses completely reciprocal concept to compression. In other words, at first, read data from compression file to form original compression table and decrypt original file from the compression file. The decryption algorithm to form compression table from the compression file is as follows.

[Decryption algorithm to form compression table from the compression]

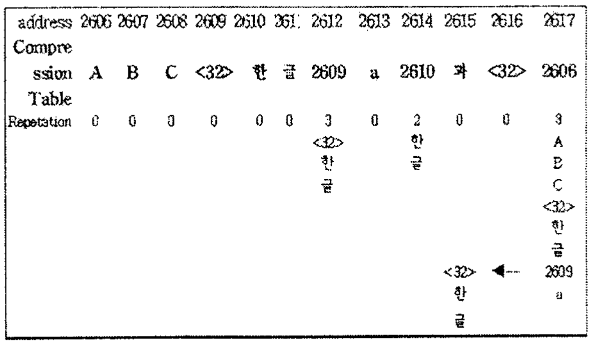
```
index = 2606;
. . .
creat_compress_table( )
{
while((ch = getc(compressed_file)) != EOF) {
  switch( ch ) {
    case 'table_address' : write table_address to prefix of TABLE;
                    write count to suffix of TABLE;
    case 'english_address': write english_address to prefix of Table;
                    write 0 to suffix of TABLE;
    case 'hangeul_address': write english_address to prefix of Table;
                   write 0 to suffix of TABLE;
  }
}
}
```

The algorithm to decrypt into original file before compression from the compression table is as follows.

[The algorithm to decrypt into original file before compression from the compression table]

```
current_index = 2606;
. . .
decompress( )
{
while( read record in TABLE[current_index] <= LIMT) {
  if(suffix of TABLE[index] == 0 )
    write prefix of TABLE[current_index] to input_file;
  else {
      save current_index to varialble tmep;
      call decompress(current_index) until reach to temp;
}
}
```

<Fig. 3>The process of decryption from the compression table



address	2606	2607	2608	2609	2610	2611	2612	2613	2614	2615	2616	2617
Compre ssion Table	A	B	C	<32>	한	글	2609	a	2610	과	<32>	2606
Repetation	0	0	0	0	0	0	3	0	2	0	0	3
							<32>		한			A
							한		글			B
							글					C
												<32>
												한
												글
										<32>	◀--	2609
										한		a
										글		

IV. Conclusion

Existing WanSeongHyeong Hangeul compression algorithm extends and improves LZW algorithm to fit into processing Hangeul.

To solve the disadvantage of existing Method 1 and Method 2, this study did not input next character in suffix of the compression table and used counter to calculate the number of repeated character within the compression table. Finally the following conclusion is drawn.

(1)By inputting the address of alphabet or Hangeul only in high-order suffix of compression table, the problem of Method 2 which divides Hangeul into high-order byte and low-order byte is solved and compression efficiency is increased.

(2)By using low-order suffix of compression table as the counter of repeated character within compression table, the problem of increasing overall size of compression file is removed, compared Method 1 and 2.

(3)When only one character registered after address 2606 after initialized table corresponds, relevant address is not used and initialized address is registered in compression table. It decreases address size in creating compression file.

In conclusion, ELZW algorithm developed in this study has excellent compression rate, compared to existing methods to compress WanSeongHyeong Hangeul and alphabet/Hangeul mixed text file. In reality, WanSeongHyeong Hangeul has been increased by 8% in compression rate, compared to commercialized LZW1 compression program and by 15%, compared to LZW2.

References

[1] Jin-uk Jung, "*Compressing Coding & secret Coding on Real-time Writing a good paper,*" KOFST. on General Writing, Vol. 17, no. 6, Nov. 2000

[2] Sung-jo Han, *Compressing Coding Method secret for WanseongHyeong Hangeul,* KOFST. On General Writing, Vol. 20, No.9, Sep 1993.