DH-LZW: LOSSLESS DATA HIDING IN LZW COMPRESSION

Hiuk Jae Shim, Jinhaeng Ahn, and Byeungwoo Jeon

School of Information and Computer Engineering, Sungkyunkwan University
300 Chunchun-dong, Jangan-gu, Suwon, 440-746, KOREA
waitnual@ece.skku.ac.kr, skajh@ece.skku.ac.kr, bjeon@yurim.skku.ac.kr

ABSTRACT

LZW is one of the well-known lossless compression methods. Since it has several remarkable features such that coding is simple, prior analysis on source is unnecessary, and the whole code table is not sent to its decoder, LZW is widely used in many applications. GIF, TIFF, and PDF (compressing images) are good examples of them. In spite of its reputation on compression, however, few data hiding methods are directly applied to LZW itself. This may be due to few redundancies remained in losslessly compressed data: therefore there is not enough available room for data hiding. The existing methods are lossy, however, lossless approach is preferred to the lossy one. In this paper, we propose the DH-LZW method that embeds data to source data in a lossless manner. Through this paper, modifiable elements of LZW for data hiding are introduced and a way to handle them is proposed. Experiments show very promising results.

1. INTRODUCTION

A lot of data hiding methods have been developed as a mean of secret data communication. Accordingly, numerous techniques have been proposed in the name of either steganography or watermarking, which all belong to data hiding techniques in wide sense. In data hiding, the most common way is a target-based method which means that a specific target such as domain (frequency, time, or spatial) is pre-determined before developing a hiding method. In this paper, we are especially interested in developing a *lossless* data hiding method that can be generally applied to many common lossless compression applications.

Among several approaches in data or image compression, LZW is a well-known technique. Since it refers to a dictionary storing single and their combined symbols, LZW is classified as a dictionary-based technique. It is different from other LZ-family such as LZ77 in defining and handling the matching window. However, they still share common characteristics in that while reading a new symbol (and decompressing codes by decoders), the encoder and its decoder both construct a code table (i.e., dictionary), therefore, there is no need to convey the updated code table to decoder. In addition, the LZ-family is a kind of universal algorithm - data is compressed without any prior knowledge such as the probability density function of source. LZW has been widely used in many applications, for instance, in GIF and TIFF, and PDF writer (in compressing image). However, proposed are few methods which embed data directly into LZW compressed data. Rather, a number of data hiding methods have been applied to GIF than LZW itself. The reason is that there rarely is redundancy in LZW compressed data. As a consequence, there is very few data hiding methods built directly on LZW.

There are two kinds of information which can undergo modification in order to hide data in GIF: one is the color definition of palette, and the other is the color index of palette. The parity of palette index can be modified to embed data [1], [2]. Niimi et al. has introduced one way to apply BPCS method to palette-based images. Since BPCS alters one channel of color index, the modified value may not be included in 256 colored palette. In this case, newly generated color definition replaces that of the color index which has the smallest distance from the new one [3]. Ogihara et al. proposed a method which is quite different from others in that they utilized both the value and length of color index [4]. While previous data hiding methods modify palette indexes, value of palette, or color components of source, the compression algorithm itself of LZW has no reason to be left out in modification for data hiding. One way is to match the parity of a prefix code index and hiding data: if they match, then the code is unchanged; otherwise, the value of the last symbol in codeword is changed in order to find another parity-matching code in the given dictionary. If the matched code is not found in the dictionary, the code is reduced by one symbol to find a match. One disadvantage of this algorithm is the distortion to the original data during data embedding

0-7803-8554-3/04/$20.00 ©2004 IEEE.

procedure. After data hiding, no severe data distortion may occur in case of 256 color indexes; however it can pose a visually serious problem especially when the original image is represented by only small number of colors such as 32, 16, etc. In this paper, we propose DH(data hiding)-LZW algorithm which causes no distortion to source and is applicable not only to GIF algorithm, but to more wide scope of applications employing the LZW algorithm. Therefore the proposed DH-LZW can be applied to any kind of data only if it is compressed by LZW. The proposed method is described in Section 2, and in Section 3, its experimental results will be shown. We summarize the whole procedure and draw conclusion in Section 4.

2. DATA HIDING IN LZW

2.1. Overview of DH-LZW

The compression process of LZW in GIF is relatively simple, not like statistical strategies such as Huffman, or the arithmetic coder, etc. Without any need to investigate the whole probability density of source, LZW is performed as it encounters a new symbol. Moreover, the code table, or the dictionary is constructed dynamically at the decoder side as well as the encoder side, which means that the same procedure of rebuilding their own dictionary is performed at both encoder and decoder. Since dynamical dictionary updating precludes sending additional information describing changes to the dictionary, it gives us a new possibility of considering a virtual dictionary as a target domain where some data can be embedded. It is one of the main differences that a virtual domain is different from other domain or space, for example, other data hiding schemes usually deal with general domain such as spatial, DCT, DWT domain, etc., and which indicate that before designing a data hiding encoder, we have to fully understand the domain where data is about to be embedded. The main concern of conventional schemes is how to understand and utilize the characteristics of their domain to embed data as much as, or as robust as possible. While what we consider here is the domain which exists virtually, in other words data is embedded in a virtual space, or dictionary itself. From this point of view, our data hiding scheme is distinguished from others. There are, moreover, additional advantages when we take a virtual domain into account instead of widely used domains. One of them is that decoded output is exactly the same as the original. Even though we modify some features from the original source in order to embed data, they exist only virtually, consequently the resulting data from processing becomes unchanged at all. That is, the proposed method belongs to the category of *lossless* data hiding which does not cause any distortion during processing the source.

2.2. Procedure of DH-LZW

As described in Section 1, there are two different elements in GIF encoder which may undergo modification to embed messages. One is the symbol definition, and the other is the length of the symbol, where length means how many single letters are joined together to form the symbol. Modifying the first one in data hiding surely causes distortion to source data, however, changing the second information does not bring distortion if one can actually find a possible way to control the second element properly. Change of the symbol length actually refers to reducing the length by defining a new symbol by detaching the last letter of the original symbol. When the length is reduced by 1, there always exists a symbol which equals the "reduced symbol" since LZW compression utilize a previously updated symbol as the prefix of a new symbol about to be updated. Note that reducing the length, or, in a more general term, forced update of an already existing symbol does not bring any compliance problem to the LZW algorithm, although it may cause slight decrease in compression performance. We utilize this forced update as means for data hiding. For better understanding of the forced update, the difference between conventional LZW and DH-LZW is illustrated in Figure 1.

LZW			DH-LZW			
input	output	new symbol	input	output	new symbol	Data embedded
a	a	256=aa	a	a	256=aa	
b	a	257=ab	b	a	257=ab	
b	b	258=bb	b	b	258=bb	
a	b	259=ba	a	b	259=ba	
bb	257	260=abb	bb	257	260=abb	
aa	259	261=baa	aa	259	261=baa	
bba	260	262=abba	bba	260	262=abba	1
ba	257	263=aba	ba	257	263=aba	
bbab	262	264=abbab	bbab	260	264=abba	1
ba	258	265=bba	bab	262	265=abbab	0

Figure 1: The procedures of usual LZW and DH-LZW, input data to compress: abbabbaabbababbabbab

The proposed DH-LZW process (as shown in Figure 1) defines a THD (=3) to skip data hiding for the symbol whose length is less than THD. However, when a symbol whose length is larger than 3 occurs, data embedding becomes possible. Consequently data hiding procedure is

started in accordance with message bits. As shown in Figure 1, symbols having code indexes from 256 to 261 undergo usual updating process since the symbol length is not larger than THD, and symbol with code indexes 262, 264 and 265, where the length meets the condition of THD, DH-LZW encoder changes the symbol length in order to embed message bits. The whole encoding procedure is as follows.

Step.1: Determine a THD value which specifies the shortest symbol length to embed a message bit.

Step.2: During LZW procedure, check whether the symbol length is greater than THD value

Step.3: If a symbol whose length is larger than THD is encountered, read a bit from embedding message.

Step.4: Check whether the parity of the symbol length and the bit read match. If they match, the symbol is updated into the dictionary as it is. Otherwise, get rid of its last letter and perform a forced update with the reduced symbol. In this manner the parity is changed from 0 to 1 or 1 to 0 so that the changed parity matches with that of the message bit.

Step.5: The procedure is repeated from 2 to 4 until exhausting the message bits.

LZW				DH-LZW				
input	old code	out-put	new table entry	input	old code	out-put	new table entry	data
a		**a**		a		**a**		
a	a	a	256=aa	a	a	a	256=aa	
b	a	b	257=ab	b	a	b	257=ab	
b	b	b	258=bb	b	b	b	258=bb	
257	b	ab	259=ba	257	b	ab	259=ba	
259	257	ba	260=abb	259	257	ba	260=abb	
260	259	abb	261=baa	260	259	abb	261=baa	1
257	260	ab	262=abba	257	260	ab	262=abba	
262	257	abba	263=aba	260	257	abb	263=aba	1
258	262	bb	264=abbab	262	260	abba	264=abba	0
257	258	ab	265=bba	b	262	b	265=abbab	

Figure 2: Procedures of LZW and DH-LZW decoding

Before we outline the decoding process, it is important to show why the forced update does not affect conventional LZW decompression process. In fact, the reason is simple: although the forced update causes redundant symbols in dictionary, LZW compression process uses the earliest symbol as the prefix of the new symbol during searching the match, and in LZW decompression, decoder reads a symbol and updates it in exactly the same manner with compression step. That is, the LZW decoder imitates the encoder's process of updating dictionary, which makes LZW decompression feasible, regardless of the redundant symbol. The difference between a usual LZW decoder and the DH-LZW decoder is illustrated by an example in Figure 2. The columns in the table denoted by "input" are from the results of Figure 1. From Figure 2, what we can observe is that the right side DH-LZW decoder decompresses the input data stream in the exactly same manner as a usual LZW decoder, accordingly the retrieval of hidden data is the only difference.

The extraction of embedded data in DH-LZW decoding is as follows.

Step.1: Process LZW decompression of the compressed data stream.

Step.2: During the process, if we encounter a symbol whose length is larger than THD, extract the parity of its length. This parity bit is the retrieved embedded bit. Similarly, when the symbol length is equal to THD, we check if the same symbol with different index exists. If there is more than one, then the symbol is regarded as a forced updated one, therefore the parity of the length is read also as an embedded bit. Otherwise the extraction step is skipped.

Stpe.3: Repeat the procedure until all the compressed data bits are exhausted.

3. EXPERIMENTAL RESULTS

The proposed algorithm is not limited to image compression such as GIF, therefore simulation is performed using both text and image data sets. Four text data are used as sample texts: "news.txt" is a small sized journal, "mc.txt" is also a journal but with bigger size, and "paper.txt" is a research paper with small size, and "file.txt" is an intentionally written text which contains a lot of artificially repeated phrases. For the simulation of image, four well-known images such as lena, peppers, splash, and baboon images are considered. They have the same file size in order to observe the different effects about each unique feature of their own. And each size of text and image files is shown in Table 1 and 2.

As shown in Table 1, in most cases of text data, file size is increased usually by 1~5%, except for the "file.txt" case. Since "file.txt" contains intentionally repeated patterns, the forced update process affects the compression performance most. The repetitive patterns are one of dominant factors influencing the LZW compression, whereas the forced update is the process which reduces the match of pattern, or makes redundancies in a dictionary. Therefore those explain why the size of "file.txt" increases up to 19.48 %.

The performance of DH-LZW on images is similar to the text data, however we may observe that the result with image is slightly better than the case with text data. This is because text data is composed of ASCII code whose

range is narrower than the range of index colors for image. In fact, the values of frequently used ASCII codes are between 0 and 128, on the contrary, the value of color index used in sample images is 0 to 255. With this fact, it is predictable in general that the symbols with longer length would occur more in text case than the image case.

The proposed DH-LZW algorithm embeds data at the cost of compression efficiency. Therefore there arise two different figures of merit. The one is the maximum amount of payload and the other one is the insensitivity in file size increment after embedding. The maximum payload size and the degree of file size increment can be controlled by setting an appropriate threshold value (THD); however, both can not be achieved at the same time. For example, as in case of peppers image, we may have the size of embedding almost 3 times larger by setting THD = 4, however the percentile of increased compression size also becomes 3 times larger. Therefore we need to determine which direction leads us to more meaningful result, in advance. While a mathematical model about payload (or capacity) is required to estimate the possible amount of embedded data, it is not that simple. Since the forced update process alters future patterns to be unpredictable ones, there is only a way of approximating the estimation so far.

Table 1: Maximum amount embedding for each sample text with THD value 4 and 5

File Name	THD	Embedded bits	Original size	LZW	DH-LZW	Increased size (%)
mc.txt	4	5580	63,829	35677	34006	4.91
news.txt	4	688	11,633	6970	6736	3.47
paper.txt	4	1928	24,799	14432	13713	5.24
file.txt	4	1392	19,970	2796	2340	19.48
mc.txt	5	2173	63,829	34720	34006	2.10
news.txt	5	186	11,633	6800	6736	0.95
paper.txt	5	794	24,799	13960	13713	1.80
file.txt	5	1218	19,970	2756	2340	17.77

Table 2: Maximum amount embedding for each sample image with THD value 4 and 5

File Name	THD	Embedded Bits	Original size	LZW	DH-LZW	Increased size (%)
Lena	4	849	65,536	58923	58649	0.467
Splash	4	3551	65,536	46161	45150	2.239
Peppers	4	6310	65,536	36909	35221	4.793
Baboon	4	1714	65,536	54373	53889	0.898
Lena	5	196	65,536	58733	58649	0.143
Splash	5	1155	65,536	45482	45150	0.735
Peppers	5	2370	65,536	35861	35221	1.817
Baboon	5	402	65,536	53943	53889	0.100

4. DISCUSSIONS

We have introduced a data hiding method in LZW compressed data. The basic concept of the proposed method, DH-LZW, is very simple compared to other conventional techniques. A THD value is defined first, and then the symbol length is utilized during the LZW compression process. The merits of DH-LZW are such that it is a lossless data hiding method, and is compatible with the conventional LZW decoders. Therefore data embedded DH-LZW bit stream can be decompressed by the general LZW decoders without any conformance problems, and when we need to extract the hidden massage, the embedded message can be retrieved only by DH-LZW decoder.

The proposed DH-LZW algorithm is a data hiding technique that embeds message data into source in a lossless manner. Since LZW compresses source with dynamically constructed dictionary, it is reasonable to handle the dictionary instead of source itself. The dictionary can be called as a virtual domain due to its implicit availability to real world processing. Another interpretation of embedding data in the virtual domain can be that it imposes additional redundancy on the dictionary of LZW. That is, as we embed messages, the efficiency of LZW compression would be decreased. Therefore the appropriate analysis of the relationship between the amount of hiding data and compression efficiency is required and it will be our next research topic.

REFERENCES

[1] Fridrich, J., and Du Dui, “A new steganographic method for palette-based images,” IS&T PICS Conference, pp.285–389, 1999.

[2] Fridrich, J., and Du Dui, ”Secure Steganographic Method for Palette Images,” 3rd Int. Workshop in Information Hiding, 1999.

[3] Niimi, M., Esaon, R.O., Noda, H., and Kawaguchi, E., “A BPCS Based Steganographic Method for Palette-Based Images Using Luminance Quasi-Preserving Color Quantization,” Proceedings of Pacific Rim Workshop on Digital Steganography 2002, pp. 84-92, Kitakyushu, Japan, July 11-12, 2002.

[4] Ogihara, T., and Kaneda, Y., “Data Embedding into Compressed Data of GIF Images,” Proceedings of Pacific Rim Workshop on Digital Steganography 2003, Kitakyushu, Japan, July 3-4, 2003.