IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received April 13, 2017, accepted May 12, 2017, date of publication May 16, 2017, date of current version June 28, 2017.

Digital Object Identifier 10.1109/ACCESS.2017.2704900

Novel Error Detection Algorithm for LZSS Compressed Data

BEOM KWON, MYONGSIK GONG, AND SANGHOON LEE, (Senior Member, IEEE)
Department of Electrical and Electronic Engineering, Yonsei University, Seoul 120-749, South Korea

Corresponding author: Sanghoon Lee (slee@yonsei.ac.kr)

This work was supported in part by the research fund of the Signal Intelligence Research Center supervised by Defense Acquisition Program Administration and in part by the Agency for Defense Development of Korea.

ABSTRACT Conventional error detection schemes, such as the repetition code, parity bit, and Hamming code, have been used to detect bit errors in data. These conventional schemes require the insertion of additional bits to detect bit errors, but the code rate decreases in proportion to the number of additional bits. In order to avoid this problem, in this paper, we introduce three special bit patterns in Lempel–Ziv–Storer–Szymanski (LZSS) compressed data. In addition, based on the three bit patterns, we propose a novel error detection algorithm for LZSS compressed data, which does not need to use additional bits to detect bit errors. In the simulation, it is demonstrated that the compression ratio and running time of the proposed algorithm are better than those of the conventional schemes, such as repetition code, parity bit, and Hamming code. In addition, it is shown that when more than/equal to seven bit errors occur, the proposed algorithm nearly always detects the presence of errors in the LZSS compressed data.

INDEX TERMS Error detection, Lempel-Ziv-Storer-Szymanski (LZSS), lossless data compression.

I. INTRODUCTION

Recently, a deluge of data from both the Internet and sensors has led to an increasing demand for efficient data storage and transmission [1]–[3]. To this end, several data compression methods have been studied. These data compression methods can be categorized into two groups: lossy and lossless compression. Lossy compression is commonly used in applications such as image, video, and audio compression, where some loss of information is acceptable [4], [5]. In contrast, lossless compression is mainly used in text and deoxyribonucleic acid (DNA) data compression, where any loss of information is unacceptable [6], [7].

Abraham Lempel and Jacob Ziv, who were pioneers in the field of lossless data compression, developed what we call the Lempel-Ziv-77 (LZ77) algorithm in 1977 [8]. Many variants of the LZ77 algorithm have since been developed [9]–[12]. One of the most popular derivatives of the LZ77 algorithm is the Lempel-Ziv-Storer-Szymanski (LZSS) algorithm [10], which was proposed in 1982 by James Storer and Thomas Szymanski. To date, the LZSS algorithm has been used in various file archiving programs such as ARJ, LHA, PKZip, RAR, and ZOO. In addition, the LZSS algorithm has been used in text [13] and heart sound compression [14], parallel processing for general-purpose computing on graphics processing units (GPGPU) [15], and a 5G-enabled internet of things (IoT) gateway design [16].

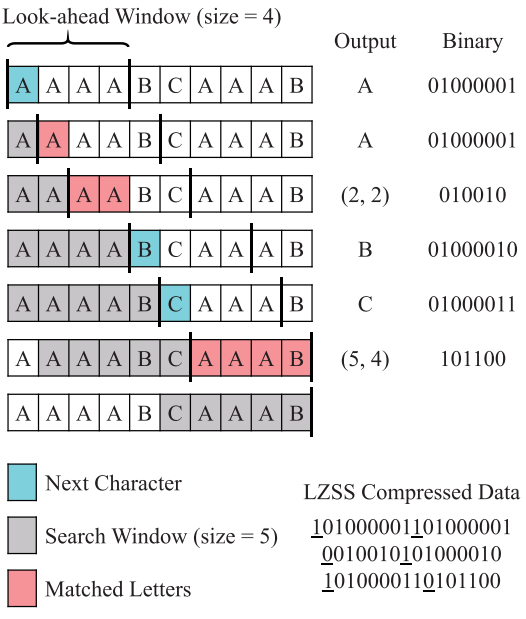

FIGURE 1. Example of lossless data compression using LZSS algorithm when text data stream of "AAAABCAAAB" is input. The sizes of the look-ahead and search windows are set to 4 and 5, respectively. The predefined minimum length *M* is set to two. The underlined digits indicate one-bit flags.

Fig. 1 shows an example of text data compression using the LZSS algorithm. Look-ahead and search windows are

utilized in the LZSS algorithm. When a text data stream is input to the LZSS algorithm, the algorithm finds the longest matching length of letters stored in the look-ahead and search windows. If the longest matching length is greater than or equal to the predefined minimum length M, the algorithm outputs (d, m), where d is the distance between the start of the matched letters in the search window and the end of the search window, and m is the matching length. If the longest matching length is less than M, the algorithm outputs the first letter l stored in the look-ahead window. When the algorithm outputs (d, m) or l, the look-ahead and search windows move by m or 1, respectively. The processes discussed above are repeatedly performed until the look-ahead window becomes empty. In addition, after the processes are completed, each output is converted into a binary form. Notably, because the LZSS algorithm outputs (d, m) or l, depending on the longest matching length, it uses one-bit flag f to signify whether the next bits represent (d, m) or l. If the algorithm outputs (d, m) (l), f is set to 1 (0). In such a case, f is inserted in front of the corresponding binary form, as shown in Fig. 1. Therefore, LZSS compressed data are stored or transmitted according to the structure shown in Fig. 2.

f	l	f	l	$\cdots$	f	d	m	f	l	$\cdots$	f	d	m

FIGURE 2. Structure of LZSS compressed data. f is inserted in front of l or (d, m). Therefore, LZSS compressed data consists of several "f, l" and "f, (d, m)" pairs.

II. RELATED WORKS AND MOTIVATION

Unfortunately, in practical applications, errors can occur in LZSS compressed data for a variety of reasons. For example, dirt on the storage media can cause errors during a memory write or read operation. In addition, errors in LZSS compressed data can be generated by unpredictable interference and noise during transmission.

In order to detect such errors, error detection and recovery methods for LZSS compressed data have been studied [17]. The proposed method in [17] encodes an error sensitive part of the LZSS compressed data by using unary coding. In addition, the method copies the encoded part, and moves it to the beginning of the LZSS compressed data. Then, the method inserts a synchronization sequence into the LZSS compressed data. The errors in the error sensitive part are detected by searching the synchronization sequence, and are recovered by using a copy of the part. However, the proposed method increases the size of LZSS compressed data, and has a limitation that it cannot detect bit errors which occur outside of the part. In addition, only the compression ratio and error recovery capability performances were provided. Moreover, since the performance comparison with other related methods is not found in [17], the effectiveness of the proposed method remains unverified.

In [18], an unequal error protection scheme for LZSS compressed data has been proposed. In addition, towards this goal, a novel structure, which requires an insertion of additional redundancy, for the output data of the LZSS algorithm is introduced in [18]. However, in [18], the proposed scheme is only focused on minimization of error propagation during the decoding process of the LZSS algorithm. In addition, it is stated that error detection is performed by using the Reed-Solomon code [19], [20]. However, the error detection performance and performance comparison with other schemes are not provided.

For error detection in LZSS compressed data, conventional error detection schemes including repetition code, parity bit, and Hamming code [21] can be also used. However, these conventional error detection schemes also require the insertion of additional bits for error detection. Therefore, if the conventional error detection schemes are used for LZSS compressed data, the code rate decreases. The code rate is defined as

$$\text{code rate} = \frac{L_c}{L_c + L_a}, \tag{1}$$

where L_c is the bit length of a compressed data and L_a is the total bit length of additional bits, which are inserted in the compressed data for error detection of the compressed data.

This led us to search for a means to detect errors in LZSS compressed data without the need to insert additional bits. Toward this goal, we investigated the output binary sequences generated by the LZSS algorithm. During this investigation, we found three unique patterns in the LZSS compressed data. In addition, we realized that these three patterns could be utilized as error check conditions. In this paper, based on the three conditions, we propose an error detection algorithm for LZSS compressed data under the assumption that the receiver know the type of the compression algorithm in advance. The proposed algorithm detects the presence of errors in data without the need to insert additional bits. A detailed description of these three conditions and the proposed algorithm is presented in the next section.

III. PROPOSED ERROR DETECTION ALGORITHM FOR LZSS COMPRESSED DATA

Let H and S be the sizes of the look-ahead and search windows, respectively. In the LZSS algorithm, the binary code lengths for d and m are the same. In this paper, d and m are each encoded with L bits. Then, the total length of (d, m) is $2L$. In addition, L is determined to satisfy the condition given by

$$2^{L-1} \leq S < 2^L. \tag{2}$$

As shown in Fig. 1, if S is set to 5, the L that satisfies the condition in (2) is determined to be 3. Therefore, in this case, the binary code of (d, m) is represented with 6 bits. In contrast, regardless of L, l is encoded with 8 bits based on the American standard code for information interchange (ASCII).

Let P be the set of (d, m) pairs in the LZSS compressed data. In the description of the three error check conditions, we use the expression (d_i, m_i), where $i \in P$, to distinguish

each (d, m) pair in the LZSS compressed data. Then, the three conditions for the LZSS compressed data can be described as follows:

- d_i is greater than or equal to m_i because the number of matched letters is upper bounded by the distance between the start of the matched letters in the search window and the end of the search window. This condition can be represented as follows:

$$d_i \geq m_i, \quad \forall i \in P. \tag{3}$$

- Based on its definition d_i, d_i is less than or equal to the size of the search window S, as follows:

$$d_i \leq S, \quad \forall i \in P. \tag{4}$$

- Based on its definition m_i, m_i is less than or equal to the size of the look-ahead window H, as follows:

$$m_i \leq H, \quad \forall i \in P. \tag{5}$$

If no errors occur in the LZSS compressed data, the LZSS compressed data must satisfy the three conditions in (3)-(5). In addition, this means that if at least one of the three conditions is not satisfied, there exist some errors in the LZSS compressed data. Therefore, the three conditions can be utilized for detecting the presence of errors in the LZSS compressed data.

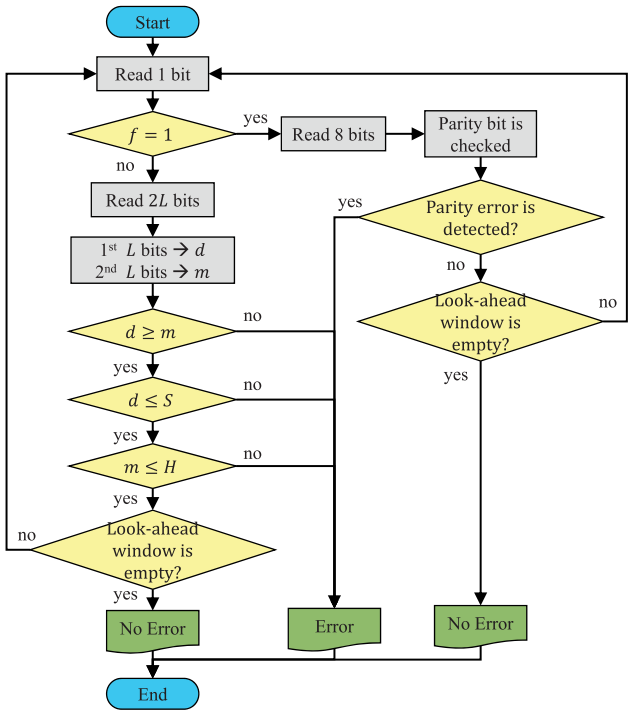

FIGURE 3. Flow chart of proposed algorithm.

Fig. 3 shows the flow chart of the proposed error detection algorithm. Because the LZSS algorithm uses a one-bit flag to signify whether the next bits represent (d, m) or l, the proposed algorithm reads one bit first in the LZSS compressed data. If the value of the flag f is 1, this means that the next eight bits represent the binary code of l. As previously described, l is encoded with 8 bits based on ASCII. In addition, only 7 out of 8 bits are used to contain the information of l, while the remaining bit is used as a parity bit. Therefore, in the proposed algorithm, when the value of f is 1, parity bit checking is performed to detect the presence of errors in the LZSS compressed data. In contrast, if the value of f is 0, the algorithm reads $2L$ bits, which represent the binary code of (d, m). The first L out of $2L$ bits is the binary code of d, and the remaining L bits are the binary code of m. After reading $2L$ bits, the algorithm checks whether d and m satisfy the three conditions in (3)-(5). These procedures are repeatedly performed until the look-ahead window becomes empty. During the procedure, if at least one of the three conditions is not guaranteed or a parity error is detected, the algorithm determines that some errors exist in the LZSS compressed data.

TABLE 1. Simulation parameters.

Parameter	Explanation/Assumption
Size of look-ahead window	$H = 18$
Size of search window	$S = 2^{16}$
Binary code length for d and m	$L = 17$

IV. SIMULATION RESULTS

In this section, we evaluate the performance of our proposed algorithm. Towards this goal, we use a desktop computer with 2.60 GHz CPU and 256 GB RAM memory. In addition, we utilizes text and data files from the two publicly available databases, namely the Calgary and Canterbury corpora [22], [23]. By referring to [24], we set the simulation parameters as shown in Table 1. To benchmark the proposed algorithm, the following three conventional error detection schemes, which are implemented by using MATLAB R2016a, are simulated:

- *Repetition code*: The main idea of this scheme is to just repeat the bit sequence r times and check whether the repeated bit sequences are equal to each other. The code rate of this scheme with r repetitions becomes $1/r$.
- *Parity bit*: In this scheme, the presence of errors is determined by the number of 1-bits in the bit stream, including the parity bit. The value of the parity bit is set to either "1" or "0" to maintain an odd number of 1-bits in the bit stream, including the parity bit. If the number of 1-bits in the bit stream, including the parity bit, is even, it is determined that a bit error occurs. If a parity bit is inserted at every n bits, the code rate of this scheme becomes $n/(n+1)$.
- *Hamming code* [21]: To determine the presence of errors in the bit stream, this scheme uses multiple parity bits. The Hamming code with h parity bits can handle a bit stream with a length of $2^h - h - 1$. The code rate of the Hamming code with h parity bits becomes $(2^h - h - 1)/(2^h - 1)$.

TABLE 2. Compression results using the LZSS algorithm for the Calgary corpus.

File name	Uncompressed size (bytes)	Compressed size (bytes)									
		$M=1$	$M=2$	$M=3$	$M=4$	$M=5$	$M=6$	$M=7$	$M=8$	$M=9$	$M=10$
bib	111,261	72,134	69,371	64,607	60,947	**59,770**	59,829	60,868	62,885	65,478	68,060
book1	768,771	586,335	580,371	566,105	542,973	**526,102**	527,187	550,355	589,537	637,194	688,107
book2	610,856	396,884	387,972	372,834	357,611	**348,693**	348,849	358,944	377,599	402,138	430,246
news	377,109	297,464	286,739	263,285	245,149	**238,827**	241,655	250,439	262,685	276,915	291,216
paper1	53,161	41,201	38,796	35,793	33,744	**32,984**	33,269	34,470	36,209	38,205	40,374
paper2	82,199	60,851	58,823	55,777	53,151	**51,629**	51,802	53,590	56,374	60,059	63,948
progc	39,611	32,076	29,251	26,455	24,888	**24,399**	24,696	25,622	26,684	28,048	29,563
progl	71,646	39,837	37,880	35,562	34,004	**33,497**	33,696	34,492	35,709	37,134	38,996
progp	49,379	28,525	26,361	24,483	23,407	**23,116**	23,399	23,860	24,610	25,484	26,411

TABLE 3. Compression results using the LZSS algorithm for the Canterbury corpus.

File name	Uncompressed size (bytes)	Compressed size (bytes)									
		$M=1$	$M=2$	$M=3$	$M=4$	$M=5$	$M=6$	$M=7$	$M=8$	$M=9$	$M=10$
alice29	152,089	104,668	102,478	98,815	95,029	**92,626**	93,005	95,886	101,334	108,199	115,691
asyoulik	125,179	97,202	95,483	92,073	87,777	**84,889**	85,184	89,141	95,230	102,315	109,678
fields	11,150	8,378	7,153	6,467	6,211	**6,149**	6,181	6,354	6,534	6,795	7,039
grammar	3,721	3,594	2,840	2,484	2,336	**2,314**	2,336	2,385	2,487	2,594	2,675
lcet10	426,754	273,470	268,276	260,051	251,224	**244,635**	244,133	250,931	264,535	282,606	303,939
plrabn12	481,861	367,050	364,489	356,455	343,459	**330,880**	331,778	344,135	370,449	404,315	438,665
xargs	4,227	4,930	3,965	3,381	3,160	**3,127**	3,172	3,224	3,334	3,463	3,551

A. COMPRESSION RATIO PERFORMANCE

In order to evaluate the performance, we compress each corpus by using the LZSS algorithm with different M values. Tables 2 and 3 represent the compression results using the LZSS algorithm for the Calgary and Canterbury corpora, respectively. Then, we calculate the compression ratio, which is defined as

$$\text{compression ratio} = \frac{\text{compressed size}}{\text{uncompressed size}}. \tag{6}$$

TABLE 4. Length of the binary code of the matched letters for $M = 4, 5, 6$ according to the number of the matched letters.

Number of the matched letters	$M=4$	$M=5$	$M=6$
3	24 bits	24 bits	24 bits
4	34 bits	**32 bits**	**32 bits**
5	**34 bits**	**34 bits**	40 bits
6	34 bits	34 bits	34 bits

Fig. 4 shows the compression ratio of the LZSS algorithm. As shown in Fig. 4, it is observed that the optimal value of M, which minimizes the compression ratio, is 5. The reason of this can be found in Table 4. As described in Table 1, d and m are encoded with $L = 17$ bits, respectively. Therefore, the total length of the binary code of (d, m) is 34 bits. Considering that l is encoded with 8 bits regardless of L, it is not preferable to set the value of M to 4 or less. If the value of M is set to 4, the matched four letters are encoded with 34 bits in a binary form of (d, m). However, if the value of M is set to 5 or 6, the matched four letters are encoded with 8 bits, respectively. Then, the total length of the binary code of the four letters becomes 32 bits. Therefore, in this case, the letters can be further compressed by setting the value of M to 5 or 6 as shown in Table 4. Similarly, when the number of the matched letters is 5, the lengths of the binary code of the matched five letters for $M = 4, 5, 6$ become 34, 34, and 40 bits, respectively. In this case, in order to compress the matched five letters more efficiently, it is desirable to set the value of M to 4 or 5. In addition, the lengths of the binary code of the matched letters for $M = 4, 5, 6$ are the same when the number of the matched letters is any value other than 4 and 5 as shown in Table 4. Therefore, the optimal value of M is 5 for $L = 17$.

TABLE 5. Compression ratio of the LZSS algorithm with $M = 5$ for the Calgary corpus after each error detection scheme is applied.

File name	**Proposed**	*Repetition code* $(r=2)$	*Parity bit* $(n=4)$	*Hamming code* $(h=6)$
bib	0.5372	1.0744	0.6715	0.5736
book1	0.6843	1.3687	0.8554	0.7307
book2	0.5708	1.1417	0.7135	0.6095
news	0.6333	1.2666	0.7916	0.6762
paper1	0.6205	1.2409	0.7756	0.6625
paper2	0.6281	1.2562	0.7851	0.6707
progc	0.6160	1.2319	0.7700	0.6577
progl	0.4675	0.9351	0.5844	0.4992
progp	0.4681	0.9363	0.5822	0.4999

To detect errors in LZSS compressed data, the conventional schemes insert additional bits into the LZSS compressed data, as previously mentioned. However, our algorithm uses no additional bits to detect the presence of errors in LZSS compressed data. As a result, the length of the output bit stream of each of the other schemes is longer than that of our proposed algorithm. Tables 5 and 6 show the compression ratio of the LZSS algorithm with $M = 5$ for the Calgary and Canterbury corpora after each error detection scheme is applied. As shown in Tables 5 and 6, applying the conventional schemes to the LZSS compressed data leads to a

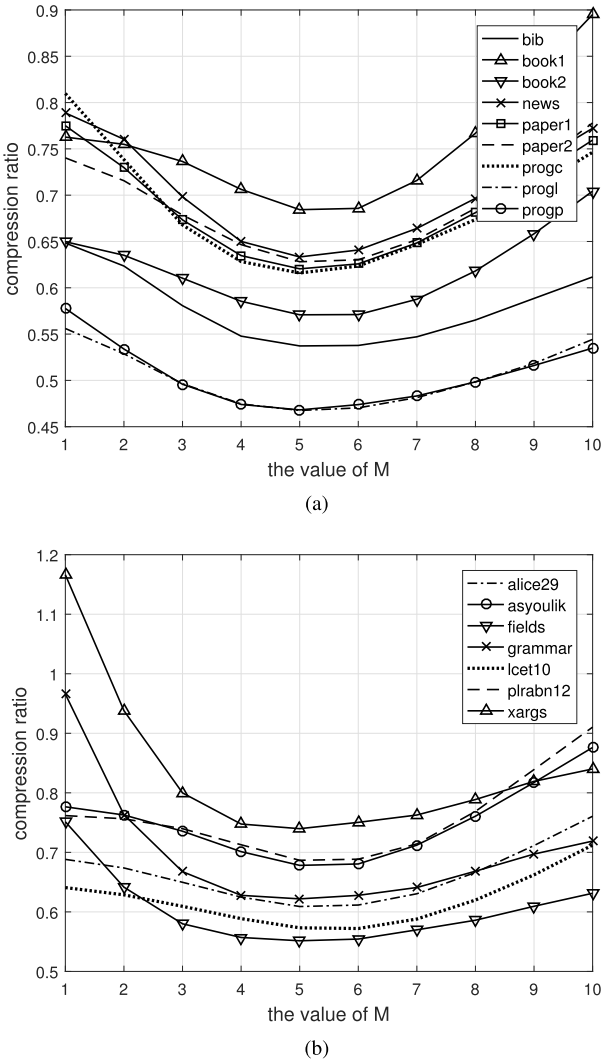

FIGURE 4. Compression ratio of the LZSS algorithm for (a) the Calgary and (b) Canterbury corpora.

TABLE 6. Compression ratio of the LZSS algorithm with $M = 5$ for the Canterbury corpus after each error detection scheme is applied.

File name	**Proposed**	*Repetition code* $(r = 2)$	*Parity bit* $(n = 4)$	*Hamming code* $(h = 6)$
alice29	0.6090	1.2180	0.7613	0.6503
asyoulik	0.6781	1.3563	0.8477	0.7241
fileds	0.5515	1.1030	0.6893	0.5889
grammar	0.6219	1.2438	0.7773	0.6640
lcet10	0.5732	1.1465	0.7166	0.6121
plrabn12	0.6867	1.3733	0.8583	0.7332
xargs	0.7398	1.4795	0.9247	0.7899

decrease in compression efficiency of the LZSS algorithm due to the insertion of additional bits. By contrast, the proposed algorithm, which does not need any additional bits, has the best compression ratio on the two databases.

B. ERROR DETECTION PERFORMANCE

In order to evaluate the error detection performance of each scheme, we define the error detection rate as

$$\text{error detection rate} = \frac{N_d}{N_t} \times 100(\%), \tag{7}$$

where N_d is the number of all correctly detected corrupted data and N_t is the total number of corrupted data.

The error detection rate of our proposed algorithm depends on both a parity check for the binary code of l and a check of the three conditions in (3)-(5) for (d, m). In the structure of the LZSS compressed data shown in Fig. 2, the percentage of l and (d, m) can change according to M. It means that the error detection rate of the proposed algorithm may change according to M.

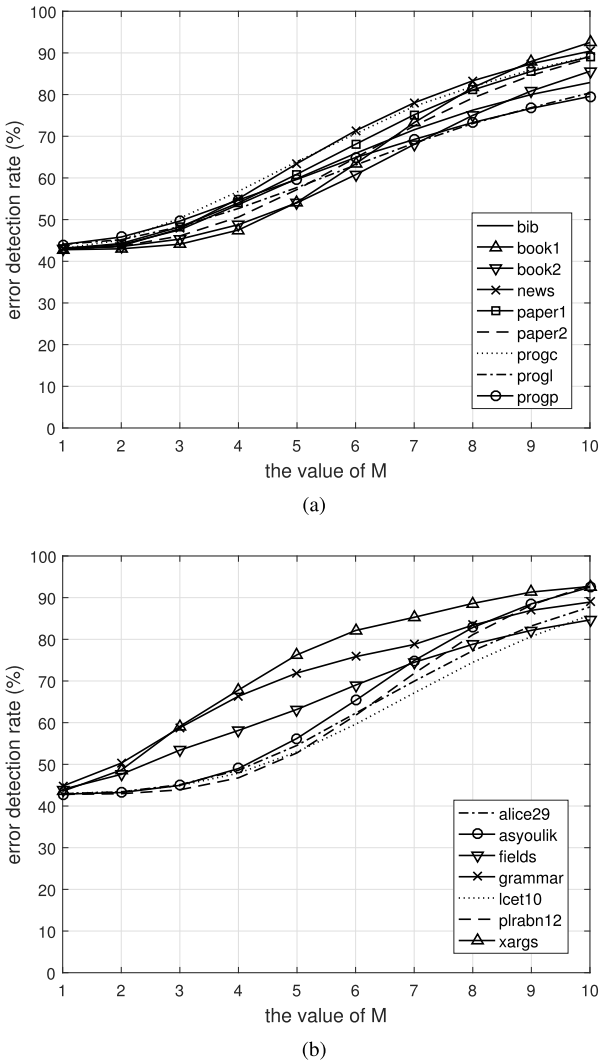

FIGURE 5. Error detection rate performance of the proposed algorithm for the Calgary and Canterbury corpora according to M.

Fig. 5 shows the error detection rate performance of our proposed algorithm for the Canterbury corpus according to M. To clearly see the performance of the proposed algorithm according to M, a single bit error is assumed in this simulation. From the figure, it is seen that the error detection rate of the proposed algorithm is improved when the value of M is set to a relatively large value. When the value of M is set to a relatively large value, the small matched letters are not encoded in a binary form of (d, m). Instead, the

matched letters are encoded with 8 bits in a binary form of l, respectively. Therefore, the percentage of l in the LZSS compressed data increases with the value of M as shown in Table 7. Accordingly, the probability that a bit error is present in the binary code of l increases as the value of M increases. In addition, if a single bit error occurs in the binary code of l, the number of 1-bits in the binary code, including the parity bit, becomes even. In this case, the parity check perfectly detects the bit error. Therefore, if the value of M is set to a relatively large value, the error detection rate of the proposed algorithm for a single bit error is improved.

TABLE 7. Numbers of f, l, and (d, m) of the LZSS compressed data for "grammar" and "xargs" files in the Canterbury corpus.

Parameter	grammar			xargs		
	N_f	N_l	N_{dm}	N_f	N_l	N_{dm}
$M=1$	878	76	802	1,182	74	1,108
$M=2$	878	308	570	1,182	371	811
$M=3$	992	571	421	1,391	832	559
$M=4$	1,135	809	326	1,619	1,207	412
$M=5$	1,274	1,003	271	1,910	1,609	301
$M=6$	1,395	1,159	236	2,153	1,922	231
$M=7$	1,508	1,296	212	2,306	2,112	194
$M=8$	1,708	1,534	174	2,513	2,357	156
$M=9$	1,890	1,746	144	2,720	2,596	124
$M=10$	2,011	1,884	127	2,845	2,737	108

N_f : # of f, N_l : # of l, N_{dm} : # of (d, m)

Fig. 6 shows the error detection rate performance of the proposed algorithm for the Calgary and Canterbury corpora according to the number of bit errors, when the value of M is set to 5. In the simulation, it is assumed that multiple bit errors occur randomly and independently in the LZSS compressed data. For performance comparison, we also evaluate the error detection rate for *Repetition code* with $r = 2$, *Parity bit* with $n = 4$, and *Hamming code* with $h = 6$. In *Repetition code* with $r = 2$, the bit sequence is repeated twice for error detection. In *Parity bit* with $n = 4$, a parity bit is inserted at every four bits, and a parity check is performed at every five bits, including the parity bit. In *Hamming code* with $h = 6$, six parity bits are inserted at every fifty-seven bits to detect a bit error.

However, we omit the results of *Repetition code* with $r = 2$, *Parity bit* with $n = 4$, and *Hamming code* with $h = 6$ for each corpus. In the simulation for Fig. 6, it is observed that the error detection rates of the conventional schemes for all the corpora are always 100%. The reason of this is as follows. In *Repetition code* with $r = 2$, the error detection may fail if a bit and its corresponding repeated bit are both flipped. However, the two bits are rarely both flipped because bit errors do not occur sequentially but randomly and independently in the simulation. As a result, *Repetition code* with $r = 2$ detects the bit error in almost all cases. However, because the code rate of *Repetition code* with $r = 2$ is smallest among the schemes, the compression ratio of *Repetition code* with $r = 2$ is also the worst as shown in Tables 5 and 6. In *Parity bit* with $n = 4$, when an even number of bits, including a parity bit, are flipped as a result of error, the scheme cannot

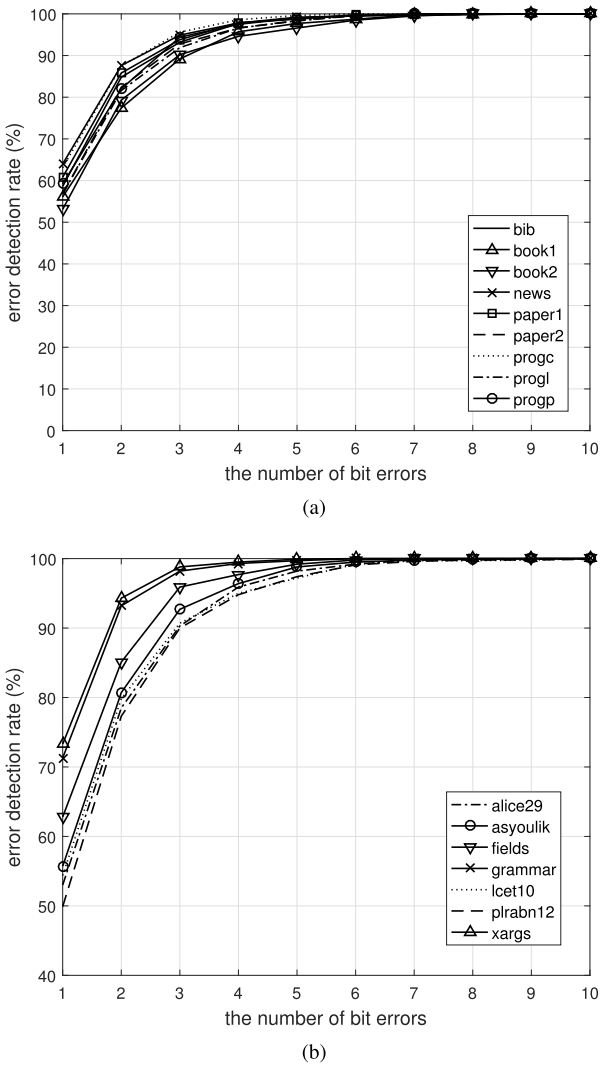

FIGURE 6. Error detection rate of the proposed algorithm for the Calgary and Canterbury corpora according to the number of bit errors, when the value of M is set to 5.

detect the presence of errors. In the simulation, it is observed that the parity check, which is performed at every five bits, nearly always detects the presence of errors because an even number of bits are rarely flipped simultaneously in the five bits. In addition, *Hamming code* with $h = 6$, which uses multiple parity bits, also nearly always detects the presence of errors in the bit stream. For this reason, in Fig. 6, we omit the results of *Repetition code* with $r = 2$, *Parity bit* with $n = 4$, and *Hamming code* with $h = 6$ for each corpus.

In Fig. 6, it is observed that the proposed algorithm falls behind from the other schemes, when the number of bit errors is smaller than or equal to 6. The reason of this is that the check of the three conditions in (3)-(5) does not always detect the bit error. For example, let us assume that the original binary code of (d, m) is "110100." The decimal values of d and m are 6 and 4, respectively. Therefore, the condition in (3) is satisfied. However, if this six-bit stream is read or received as "110101" - where the single bit error occurs in the last bit

- the decimal values of d and m become 6 and 5, respectively. However, because the condition in (3) is still satisfied, this bit error is not detected by the proposed algorithm. However, it is observed that when the number of bit errors is greater than or equal to 7, the proposed algorithm nearly always detects the presence of errors in the bit stream. In addition, as previously mentioned, *Repetition code* with $r = 2$, *Parity bit* with $n = 4$, and *Hamming code* with $h = 6$ utilize additional bits, which degrades compression efficiency of each scheme as shown in Tables 5 and 6. In contrast, our proposed algorithm does not use any additional bits for error detection. Therefore, the utility of the proposed algorithm can be greater than those of the conventional schemes, when the number of bit errors is greater than or equal to 7.

TABLE 8. Running time (in seconds) of each scheme for the Calgary corpus, when the value of *M* is set to 5.

File name	**Proposed**	*Repetition code* ($r = 2$)	*Parity bit* ($n = 4$)	*Hamming code* ($h = 6$)
bib	0.0514	0.9343	0.4328	0.1186
book1	0.4306	7.1586	3.5474	0.7528
book2	0.2688	4.7150	2.3578	0.4673
news	0.1721	3.1107	1.6369	0.3975
paper1	0.0237	0.5770	0.2231	0.0480
paper2	0.0364	0.7114	0.3517	0.0811
progc	0.0177	0.3208	0.1656	0.0418
progl	0.0280	0.4790	0.2248	0.0531
progp	0.0189	0.2850	0.1652	0.0435

TABLE 9. Running time (in seconds) of each scheme for the Canterbury corpus, when the value of *M* is set to 5.

File name	**Proposed**	*Repetition code* ($r = 2$)	*Parity bit* ($n = 4$)	*Hamming code* ($h = 6$)
alice29	0.0740	1.2141	0.6358	0.1342
asyoulik	0.0687	1.3329	0.5599	0.1170
fileds	0.0057	0.0841	0.5068	0.0109
grammar	0.0014	0.0296	0.0378	0.0054
lcet10	0.1871	3.4821	1.6631	0.3085
plrabn12	0.2752	4.5239	2.2684	0.5070
xargs	0.0022	0.0411	0.0246	0.0095

C. RUNNING TIME PERFORMANCE

To evaluate the running time performance of each scheme, we separately calculate the running time of each scheme by using the TIC and TOC functions in MATLAB R2016a. The TIC function records the current time, and the TOC function measures the time elapsed from the time recorded by the TIC function. Tables 8 and 9 show the running time of each scheme for the Calgary and Canterbury corpora, respectively. In this simulation, the value of M is set to 5. In Tables 8 and 9, it is observed that the proposed algorithm performs best on all the corpora. One of the reasons for this is that the length of the bit sequence of the compressed data for each corpus increases due to the insertion of additional bits, when the conventional schemes are applied. The running time of each scheme depends on the length of bit sequence. In general, the running time increases as the length of bit sequence increases. These results are consistent with the compression ratio performances as shown in Tables 5 and 6.

V. CONCLUSION

Existing error detection schemes, like repetition code, parity bit, and Hamming code, require the use of additional bits for error detection. Therefore, if these conventional schemes are used for LZSS compressed data, a degradation of the code rate of the compressed data occurs. In order to avoid this problem, in this paper, we proposed a novel error detection algorithm for LZSS compressed data. Toward this goal, we introduced three error check conditions that stemmed from the unique bit patterns in LZSS compressed data. These three conditions allow the proposed algorithm to detect the presence of errors in LZSS compressed data. Because our proposed algorithm used no additional bits for error detection, it could outperform the conventional schemes in terms of compression ratio and running time. The proposed algorithm fell behind from the conventional schemes when the number of bit errors is smaller than or equal to 6. However, it was demonstrated that the proposed algorithm could nearly always detect the presence of errors when the number of bit errors is greater than or equal to 7. Therefore, the utility of the proposed algorithm can be greater than those of the conventional schemes, when the number of bit errors is greater than or equal to 7.

REFERENCES

[1] H. Hu, Y. Wen, T.-S. Chua, and X. Li, "Toward scalable systems for big data analytics: A technology tutorial," *IEEE Access*, vol. 2, pp. 652–687, 2014.
[2] J. G. Wolff, "Big data and the SP theory of intelligence," *IEEE Access*, vol. 2, no. 4, pp. 301–315, 2014. [Online]. Available: http://bit.ly/1jGWXDH
[3] C. Zhu, V. C. M. Leung, L. Shu, and E. C.-H. Ngai, "Green Internet of Things for smart world," *IEEE Access*, vol. 3, pp. 2151–2162, 2015.
[4] F. Jiang, X. Ji, C. Hu, S, Liu, and D. Zhao, "Compressed vision information restoration based on cloud prior and local prior," *IEEE Access*, vol. 2, pp. 1117–1127, 2014.
[5] S. L. Chen, T.-Y. Liu, C.-W. Shen, and M.-C. Tuan, "VLSI implementation of a cost-efficient near-lossless CFA image compressor for wireless capsule endoscopy," *IEEE Access*, vol. 4, pp. 10235–10245, 2016.
[6] S. Shanmugasundaram and R. Lourdusamy, "A comparative study of text compression algorithms," *Int. J. Wisdom Based Comput.*, vol. 1, no. 3, pp. 68–76, 2011.
[7] S. Kuruppu, B. Beresford-Smith, T. Conway, and J. Zobel, "Iterative dictionary construction for compression of large DNA data sets," *IEEE/ACM Trans. Comput. Biol. Bioinf.*, vol. 9, no. 1, pp. 137–149, Jan. 2012.
[8] J. Ziv and A. Lempel, "A universal algorithm for sequential data compression," *IEEE Trans. Inf. Theory*, vol. IT-23, no. 3, pp. 337–343, May 1977.
[9] J. Ziv and A. Lempel, "Compression of individual sequences via variable-rate coding," *IEEE Trans. Inf. Theory*, vol. IT-24, no. 5, pp. 530–536, Sep. 1978.
[10] J. A. Storer and T. G. Szymanski, "Data compression via textual substitution," *J. ACM*, vol. 29, no. 4, pp. 928–951, 1982.
[11] T. A. Welch, "A technique for high-performance data compression," *Computer*, vol. 6, no. 17, pp. 8–19, 1984.
[12] R. N. Williams, "An extremely fast Ziv-Lempel data compression algorithm," in *Proc. IEEE Data Compress. Conf. (DCC)*, Apr. 1991, pp. 362–371.
[13] J. Y. Lee and K. M. Sung, "Modification of LZSS by using structures of Hangul characters for hangul text compression," *IEICE Trans. Fundam. Electron., Commun. Comput. Sci.*, vol. 79, no. 11, pp. 1904–1910, 1996.
[14] W. Qin and P. Wang, "A remote heart sound monitoring system based on LZSS lossless compression algorithm," in *Proc. IEEE 4th Int. Conf. Electron. Inf. Emergency Commun. (ICEIEC)*, Jun. 2013, pp. 109–112.
[15] A. Ozsoy, M. Swany, and A. Chauhan, "Pipelined parallel LZSS for streaming data compression on GPGPUs," in *Proc. IEEE 18th Int. Conf. Parallel Distrib. Syst. (ICPADS)*, Dec. 2012, pp. 37–44.

[16] N. Saxena, A. Roy, B. J. R. Sahu, and H. Kim, "Efficient IoT gateway over 5G wireless: A new design with prototype and implementation results," *IEEE Commun. Mag.*, vol. 55, no. 2, pp. 97–105, Feb. 2017.
[17] M. Kitakami and T. Kawasaki, "Burst error recovery method for LZSS coding," *IEICE Trans. Inf. Syst.*, vol. 92, no. 12, pp. 2439–2444, Dec. 2009.
[18] Z. C. Pereira, M. E. Pellenz, R. D. Souza, and M. A. Siqueira, "Unequal error protection for LZSS compressed data using Reed–Solomon codes," *IET Commun.*, vol. 1, no. 4, pp. 612–617, Aug. 2007.
[19] F. J. MacWilliams and N. J. A. Sloane, *The Theory of Error-Correcting Codes*. Amsterdam, The Netherlands: Elsevier, 1977.
[20] S. Lin and D. J. Costello, *Error Control Coding*. Upper Saddle River, NJ, USA: Pearson, 2004.
[21] R. W. Hamming, "Error detecting and error correcting codes," *Bell Syst. Tech. J.*, vol. 29, no. 2, pp. 147–160, Apr. 1950.
[22] T. Bell, I. H. Witten, and J. G. Cleary, "Modeling for text compression," *ACM Comput. Surv.*, vol. 21, no. 4, pp. 557–591, 1989.
[23] R. Arnold and T. Bell, "A corpus for the evaluation of lossless compression algorithms," in *Proc. IEEE Data Compress. Conf. (DCC)*, Mar. 1997, pp. 201–210.
[24] M. Kitakami and Y. Noguchi, "Error recovery method for multiple-dictionary compression method," in *Proc. IEEE 13th Pacific Rim Int. Symp. Dependable Comput. (PRDC)*, Dec. 2007, pp. 11–18.

BEOM KWON was born in South Korea in 1989. He received the B.S. degree in electrical and electronic engineering from Soongsil University, Seoul, South Korea, in 2012. He is currently pursuing the M.S. and Ph.D. degrees with the Multidimensional Insight Laboratory, Yonsei University. His research interests are in the area of wireless communication networks, computer vision, and machine learning.

MYONGSIK GONG was born in South Korea in 1993. He received the B.S. degree in electrical and electronic engineering from Yonsei University, Seoul, South Korea, in 2016. He is currently pursuing the M.S. and Ph.D. degrees with the Multidimensional Insight Laboratory, Yonsei University. His research interests are in the area of computer vision and machine learning.

SANGHOON LEE (M'05–SM'12) received the B.S. degree from Yonsei University, Seoul, South Korea, in 1989, the M.S. degree from the Korea Advanced Institute of Science and Technology in 1991, and the Ph.D. degree from The University of Texas at Austin in 2000, all in electrical engineering. From 1991 to 1996, he was with Korea Telecom. From 1999 to 2002, he was with Lucent Technologies on 3G wireless and multimedia networks. In 2003, he joined the Faculty of the Department of Electrical and Electronics Engineering, Yonsei University, where he is currently a Full Professor. His current research interests include image/video quality assessment, computer vision, graphics, cloud computing, and multimedia communications and wireless networks. He currently serves as a member in the Technical Committee of the IEEE Multimedia Signal Processing since 2016, the IEEE IVMSP Technical Committee since 2014, and the APSIPA IVM TC Vice Chair since 2016. He received the 2015 Yonsei Academic Award from Yonsei University, the 2012 Special Service Award from the IEEE Broadcast Technology Society, and the 2013 Special Service Award from the IEEE Signal Processing Society. He was the Technical Program Co-Chair of the International Conference on Information Networking 2014 and the Global 3-D Forum 2012 and 2013, and the General Chair of the 2013 IEEE IVMSP Workshop. He was an Associate Editor of the IEEE TRANSACTIONS ON IMAGE PROCESSING from 2010 to 2014. He also served as a special issue Guest Editor of the IEEE TRANSACTIONS ON IMAGE PROCESSING in 2013, and an Editor of the *Journal of Communications and Networks* from 2009 to 2015. He has been an Associate Editor of the IEEE SIGNAL PROCESSING LETTERS since 2014 and the *Journal of Electronic Imaging* since 2015 and the Chair of the IEEE P3333.1 Quality Assessment Working Group since 2011.