Received October 18, 2018, accepted January 2, 2019, date of publication January 10, 2019, date of current version January 29, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2891764

Repair and Restoration of Corrupted LZSS Files

GANG WANG, HUA PENG, AND YONGWANG TANG
National Digital Switching System Engineering and Technology Research Center of China, Zhengzhou 450002, China

Corresponding author: Gang Wang (angwzhg@yeah.net)

This work was supported by the National Natural Science Foundation of China under Grant 61572518 and Grant 61501516.

ABSTRACT Data compression and decompression have been widely used in modern communication and data transmission. But how to decompress the corrupted lossless compressed files remains a challenge. Aiming at the Lempel–Ziv–Storer–Szymanski (LZSS), a lossless data compression algorithm widely used in the field of general coding, this paper proposes an effective method to repair the errors and decompress and restore the corrupted LZSS files, and provides the theoretical basis for the method. By using the residual redundancy left by the LZSS encoder to carry the check information, the method can repair the errors in LZSS compressed data without any loss of compression performance. The proposed method neither requires additional bits nor changes coding rules or data formats. It is fully compatible with standard algorithms. That is, the data compressed by LZSS with error repair capability can still be decompressed by the standard LZSS decoder. The experimental results verify the validity and practicability of the proposed method.

INDEX TERMS Corrupted files, error repair, lossless data compression, residual redundancy.

I. INTRODUCTION

Source coding technology is used in various communication systems. The purpose of source coding is to remove redundancy and describe information with as few bits as possible [1]; thus, source coding is also called data compression. Limited communication resources require efficient data compression techniques, whereas noisy wireless channels and corrupted file systems require repair capabilities.

The lossless data compression method Lempel-Ziv-Storer-Szymanski (LZSS) and its variants are used extensively in many compression schemes (e.g. Zip, Pdf, Png, Office documents). This popular compression method works online in real time. It replaces the longest prefix of the uncompressed portion in files with a pointer to the same prefix of the already compressed portion (including the position and length of the longest prefix). However, LZSS has a main disadvantage of poor anti-error performance. Even a single error may spread and produce numerous error codes in the decoding process [2].

Error-correcting decompression technology can repair data errors and decompress corrupted data. This technology has been widely used in audio or video decoding [3], [4]. The lossless compressed files are difficult to repair because of their low redundancy. The corrupted lossless compressed files can be retrieved by backup or retransmission. There are few studies on error-correcting decompression of lossless compressed files.

With the development of the Internet technology, the amount of data on the Internet has increased dramatically, making it difficult to backup and retransmit all lossless compressed files. In addition, with the emergence of the applications such as the wireless sensor networks and the Internet of Things, data are becoming larger and more diverse. The limited storage space and bandwidth make it impossible to backup and retransmit the lossless compressed files. Therefore, it becomes really necessary to repair the errors of the corrupted lossless compressed files.

The error-correcting method for the lossless compressed files was originally designed to decompress the corrupted compressed files intercepted from wireless channels [5], [6]. The error-correcting decompression method needs to be employed to repair the corrupted intercepted compressed data in non-cooperative communication, for there is no way to get the data retransmitted.

A possible solution to this problem is to protect the compressed data by adding additional check code so that error detection and correction can be performed during the decompression process. The number of bits required to carry the check information in such systems should be as small as possible. Kwon *et al.* [7] introduced three special bit patterns in compressed data and proposed an error detection method, which required no additional bits to detect the error codes, but he failed to develop a feasible error-correcting method for the corrupted compressed file. Wang *et al.* [8] established a mathematical model for error bit detection by using compression coding rules and grammar rules, estimated the rough range of error bits, and used heuristic method to determine the exact positions of the errors and correct the errors. But the heuristic method requires grammar rules as the prior information and cannot correct the replacement errors. Kostina *et al.* [9] used the information of source state and channel state transmitted with compressed data to correct the errors of compressed data at certain bit error rate. Zhang *et al.* [10] grouped length-variable source codes and added check code to the grouped data. Klein and Shapira [11] analyzed the error propagation problem in dictionary encoding and transformed the multiple modes in dictionary encoding rule into the uniform standard codes. This could increase the decoding speed and provide stronger robustness at occurrence of mistakes. However, the error recovery problem remained unsolved. Kitakami and Kawasaki [12] encoded the error's sensitive parts such as matching length codes and marker bits in the compressed data by unary coding, and inserted the synchronous sequence into the compressed data. Pereira *et al.* [13] proposed an unequal error protection (UEP) scheme to detect errors according to the importance of each part of compressed data. Lakhani [14] adjusted the construction rules of the encoder and dictionary table and verified the data by adding redundant bits. Park *et al.* [15] used compression coding rules to detect corrupted ZIP compressed data. However, the exhaustive error correction methods can correct only 1 bit each time to ensure the speed of error correction. All of the above mentioned methods reduce the compression performance, because they integrate additional check code in the compressed data [16] instead of using the residual redundancy left by the compressed data. Moreover, these methods modify the standard algorithm. Due to changes in encoding rules and data formats, they are not compatible with standard algorithms, so the usability is compromised.

To effectively repair the errors of the LZSS compressed data, we propose a new scheme compatible with the standard LZSS algorithm, addressing the problem of the existing research that needs additional information bits to protect data and are incompatible with the standard algorithm. The proposed method is capable of error repairing. It does not change the encoding rules and data format, so the data compressed by the method in this study can still be decompressed by the standard LZSS decoder, not affecting the compression performance.

The LZSS encoder cannot completely remove the correlation of the input sequence; thus, redundancy still exists in the compressed data stream. The redundancy originates from the encoding when the pointer codeword is selected among multiple matching pointers. If the longest prefix has r matches, then $\lfloor log_2 r \rfloor$ bits can be embedded by selecting one of the r pointers. The matching multiplicity can be used for error repair once the redundancy bit is determined. Extensive test of the LZSS decoder shows that the method is fully compatible with the standard LZSS algorithm and will not affect the compression performance.

The remainder of this paper is arranged as follows. Section 2 introduces the basic principles of the LZSS algorithm. Section 3 elaborates on the two error correction methods LZSR and LZSRD. Section 4 discusses the theoretical analysis results of the LZSRD algorithm. Section 5 presents the implementation process and analyzes the experimental results.

II. BASIC PRINCIPLE OF LZSS ALGORITHM

When the LZSS algorithm reads the file and processes the data in real time, it parses the file from left to right and checks the encoded symbol sequence to determine the longest matching prefix of the string to be encoded starting from the current position [17]. The longest matching prefix is represented by a pointer, which is a codeword consisting of (position, length).

The basic principle of the LZSS algorithm is to determine the longest matching prefix $(X_i, X_{i+1}, \ldots, X_{i+l-1})$ of the current encoded string $\mathbf{S}= (X_i, X_{i+1}, \ldots, X_N)$ in the processed string $\mathbf{Z} = (X_1, X_2, \ldots, X_{i-1})$ and replace it with a pointer to the same prefix that appears before. The pointer is represented by the codeword $Y_k = (p_k, l_k)$, where p_k is the position of the longest matching prefix of the current index i, and l_k is the length of the longest matching prefix. Figure 1 shows that when the LZSS algorithm is encoding a sequence with starting position i, a phrase with starting position j and length $l = 9$ is matched with a prefix whose current starting position is i.

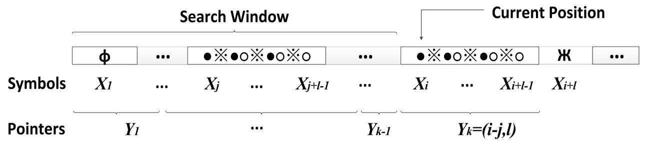

FIGURE 1. LZSS algorithm.

To avoid excessive position and length parameter values, the LZSS algorithm uses a lookup principle called a sliding window, which finds the longest matching phrase only in a fixed-size window.

Let T be the data in a finite-length symbol set A whose length is n, and $T_{[i]}(1 \leq i \leq n)$ denotes the ith symbol in T. $T[i, j]$ is used as an abbreviation for substring $T[i]\,T[i+1]\ldots T[j](1 \leq i \leq j \leq n)$, and convention $T[i, i] = T[i]$ is obtained. The prefix and suffix of T are represented as substrings $T[1, j]$ and $T[i, n]$ respectively.

Suppose the first $i-1$ symbols of the string T have been parsed out in the first $k-1$ phrases (i.e., $T_{[1,i-1]} = y_1 y_2 \ldots y_{k-1}$). To identify the kth phrase, the LZSS algorithm is used to determine the longest prefix of $T_{[i,n]}$ that matches a certain substring of $T_{[1,i-1]}$. If $T_{[j,j+l-1]}, j \leq i-l$ is a substring that matches the longest prefix, then $y_k = T_{[j,j+l-1]}$. The algorithm provides the pointer codeword $(i{-}j, l)$ and then updates the current position value from i to $i+l$.

On the basis of theory and experiment, many phrases have more than one longest matching prefix in the compressed data. Specifically, the number of longest match prefix in a sequence with a length of n generated by binary memoryless source, M_n (where p is the probability of the occurrence of 0) shall satisfy $p(M_n = j) \approx (1/h)(p^j(1-p) + (1-p)^j p)/j$, where h is the entropy of the source [18].

III. REPAIR AND RESTORATION OF LZSS FILES

The residual redundancy in source coding is introduced in the encoding process and the encoder that ignores the probability distribution characteristics of the source data [19]. The principle of the error-correcting decompression based on the dictionary compression such as LZSS is to improve the capability of error correction without reducing the compression ratio (the ratio of the compressed and uncompressed data sizes).

The residual redundancy in the compressed data stream using the dictionary-based algorithm is used to introduce validation information and provide error repair capability by embedding protection bits into the compressed data stream. This information can be used to detect and repair errors in the compressed stream.

A. LZSR ALGORITHM

The compression mechanism of LZSS must be analyzed to embed additional bits for repairing errors. The new algorithm is called LZSR algorithm, where "R" refers to error repair. The LZSS encoder cannot completely remove the correlation of the input sequence. Thus, residual redundancy still remains in the compressed data stream; this residual redundancy can be used to repair errors. More than one longest matching prefix exists for a given sequence or phrase; hence, more than one matching pointer also exists. Usually, the algorithm selects the latest pointer [20], which has the smallest position value; however, selecting another pointer does not affect the decompression process and result. If the starting position of a phrase is i from the beginning of the input sequence, and r longest prefixes exist in the sequence that exactly match position i, then the phrase has matching multiplicity r. The matching multiplicity represents the redundancy of some types. It is possible to embed additional information bits without reducing the compression ratio. The residual redundancy is generated from the process that the pointer codeword is selected from $r > 1$ possible pointers. Additional bits can be embedded in the position with $r > 1$ multiple longest matching prefixes. At most $\lfloor \log_2(r) \rfloor$ additional bits can be embedded by selecting one of the r pointer options. These additional bits can be used for various purposes, such as authentication [21] or bit error repair.

Set $T_{[1,i-1]}$, the initial part of T, has been parsed. For all $0 \leq m \leq r-1$, let $\{(p_0, l), (p_1, l), \ldots, (p_m, l), \ldots, (p_{r-1}, l)\}$, $r \geq 1$ be all the possible pointers to the longest matching prefix of $T_{[i,n]}$, where $l > 1$, $1 \leq p_m \leq i - l$. If $r = 1$, no additional information bits are embedded. When $r > 1$, one of the r pointer codewords is selected on the basis of the value of $d = \lfloor \log_2 r \rfloor$ bits in F data to be embedded. Suppose that the first t bits of F have been embedded in the previous phrase, the result of the encoding is the pointer codeword $(p_{F_{[t+1,t+d]}}, l)$. Then, T is moved to $i + l$ from the current position, and t is incremented by d. When $r > 1$ identical longest matching pointers exist, the additional bits can be encoded by reasonable selection of pointers. As shown in Fig. 2, the number of longest matching prefixes is $r = 4$. Two additional bits can be embedded by selecting one of the four matching pointers to embed the check code for error detection and repair. Selecting different pointers does not affect the decompression process; thus, the proposed algorithm is fully compatible with the standard LZSS decoder.

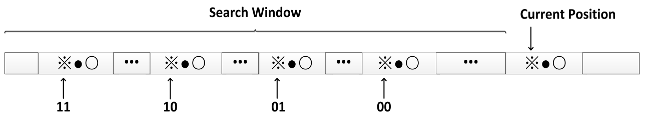

FIGURE 2. Multiplicity of the longest matching prefix.

Once the residual redundancy of the LZSS algorithm is determined, a method can be devised to repair the error by using redundant bits. The protected pointer codewords are represented by a sequence of bytes; thus, the Reed-Solomon (RS) code [22] is used to protect the data. The RS code is an error-correcting code that is widely used in digital communication and storage systems. The RS code is a BCH code and is usually expressed as RS(a, b), where a is the size of the packet containing the data and the check code, and b is the size of the payload.

The encoder collects b symbols and adds (a-b) check bits to form a packet with a length of a. If an error occurs in any bit of the packet, then a bit error is generated. The RS decoder can correct e errors in the packet, where $e = (a-b)/2$. Given a code element represented by s bits, the maximum packet length of the RS code is $a = 2^s - 1$. For example, the maximum length of a codeword with 8 bits ($s = 8$) is 255 bytes. Therefore, the RS code of $s = 8$ can be represented by RS (255, 255-2e), where 255 is the number of the bytes contained in each packet, 255-2e denotes data, and 2e is the check code. Encoding can automatically detect and correct e byte errors anywhere in the packet.

Subsequently, the manner in which the LZSR algorithm is used to embed the RS check code is introduced. The scheme initially compresses T using the standard LZSS algorithm and divides the encoded compressed data into packets with the size of 255-2e. Then, from the last one, the packets are processed in reverse order. When processing packet G_i, the encoder initially calculates the RS check code of packet G_{i+1} and embeds the RS check code into the pointer codeword of packet G_i using the multiplicity of matching. The check code of the first packet G_1 is not embedded in any packet but is stored at the beginning of the compressed file. The operation flow of the LZSR encoder processing compressed data is shown in Fig. 3.

The decompression process proceeds in the forward order. The decoder receives the sequence of pointer codewords, the first being the check code of the first packet G_1. First the input data stream is divided into packets with the size

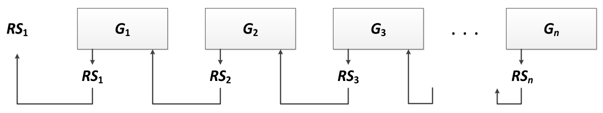

FIGURE 3. LZSR encoder (RS_n represents the check code of block G_n).

of 255-2e, and the check code is then used to repair the first packet G_1. When packet G_1 is correct, the LZSR algorithm is used to decompress it. This process not only reconstructs the first packet of the original file but also restores the bit information stored in the particular selected pointer codeword. These additional bits are collected as the check code of the second packet G_2; thus, the decoder can repair the possible errors in packet G_2, and the algorithm then decompresses packet G_2 to extract the check code of packet G_3. This process continues until all packets have been decompressed.

The RS check code of the current packet must be known prior to the decompression of the decoder. Thus, by embedding the RS check code of the current packet into the previous packet, the RS code of the current packet can be obtained as the previous packet is decompressed and the current packet is verified. Therefore, the encoder must process these packets in reverse order.

On the basis of the redundancy of the encoded data, the capability to embed bits in the multiple matching pointer determines the maximum number of errors e that can be effectively corrected in each packet when decompressing. In the LZSR algorithm, e is constant in all packets; thus, its value is limited by the packet with the least redundancy.

B. LZSRD ALGORITHM

In the LZSR algorithm, the redundancy of different parts of the data may be different; thus, the algorithm is not optimal if e takes a constant value in all encoded packets. If the redundancy of a packet in the data is extremely low, it will determine the maximum value of e in all packets. Such low-redundancy packets are usually located at the beginning of the encoded data because only few identical matches can produce redundancy at the beginning. To fully utilize the overall redundancy, the value of e that is dynamically adjusted between each packet can be used on the basis of the availability of redundant bits in each packet. In this case, the low redundancy of data only affects the error protection performance and the amount of information embedded in these parts, and the rest of the data can be protected according to its redundancy availability. Therefore, the average of e can be higher to better resist the impact of bit errors.

From the above analysis, LZSRD algorithm is proposed based on LZSR, where "D" refers to the dynamically adjusted e. The encoding process is still carried out in two steps. In the first step, the encoder not only compresses the input stream, but also determines the embedding capability of each data packet, which is used to determine the parameters of the check code. The input string X is initially encoded using the standard LZSS algorithm, and the number r of the identical longest matching prefixes that each pointer has is recorded. The data to be embedded are then divided into packets of different lengths according to the number of bits that can be embedded by the available redundancy. First, the data length of the first packet G_1 is 255-2e_1, where e_1 is used as the parameter input of the algorithm. The bits number m_i of the check code in the packet G_i is calculated on the basis of the r value of the packet G_{i-1}. m_i is calculated according to (1).

$$m_i = \sum\nolimits_{g \in G_{i-1}} \lfloor \log_2 r_g \rfloor \tag{1}$$

The check code length is $\lfloor m_i/8 \rfloor$, and it can detect and correct $e_i = \lfloor \lfloor m_i/8 \rfloor /2 \rfloor$ errors in the packet G_i.

For example, if the bits number that can be embedded on the basis of the multiple match pointer $\sum_{g \in G_1} \lfloor \log_2 r_g \rfloor$ of the first packet G_1 is 39, then the length of check code for the second packet G_2 is $\lfloor 39/8 \rfloor = 4$. On this basis, the data length of the second packet G_2 is $255\text{-}4 = 251$. In G_2, the check code can detect and correct $e_2 = 2$ errors. This process is repeated until the input data end. Finally, n packets with data length of 255-2e_n are obtained, as shown in Fig. 4.

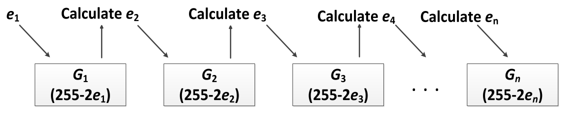

FIGURE 4. The process of calculating the embedding capability.

After all the data are sliced into packets of different lengths, the process of embedding the check bit is carried out. In the second step, these packets are processed in reverse order, and the length, 2e_i, of the information bits that can be embedded in each packet is different. When processing the data packet G_{i+1}, the encoder reads the parameter information of RS codes associated with the packet G_i, selects an appropriate RS encoder according to this information and calculates the check bit of the data packet G_{i+1}. These bits are to be embedded into the packet G_i. During this process, the last packet contains only compressed symbols and no additional information. The first packet's check bit is added to the beginning of the output stream of the encoder. As shown in Fig. 5.

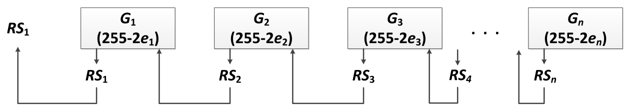

FIGURE 5. The process of embedding the check code.

The expected error correction capability, e_1, of the first packet G_1 is taken as the input parameters of the algorithm, and for all the other packets, the expected error correction capability, e_i, is based on the redundancy dynamics of its previous packet. In the LZSRD algorithm, the check code of the first packet G_1 is appended to the beginning of the encoded data. To preserve the compatibility with the standard

LZSS decoder, the check code of the first packet where $e_1 = 0$ is to be removed.

The decompression process is similar to the LZSR decoding algorithm. First, the $2e_i$ embedded check code are extracted from the packet G_{i-1}, and the packet G_i uses the check code to determine the data length and correct the errors. Then, the LZSR decoder is used to decompress the original data of the packet G_i and obtain the $2e_{i+1}$ check code of the next packet G_{i+1}, which is used to determine the data length of the packet G_{i+1} and verify the packet. This process is repeated until the last packet.

IV. ANALYSIS OF MATCHING MULTIPLICITY OF LZSRD

This section analyzes the multiplicity of matching in the LZSRD algorithm. When the data are generated by a memoryless source, the distribution characteristics of the longest matching prefix number can be estimated on the basis of the analysis result.

Let $T_{[1,n]}$ be the first n symbols generated by the source, and L is a random variable found in $T_{[1,n]}$ associated with the length of the longest matching prefix of $T_{[\mathrm{n}+1,\infty]}$. In other words, the random variable L describes the length of the longest matching prefix in the LZSRD algorithm. The variable W_n is used to indicate the number of the longest matching prefix of the sequence starting from the position n in the data, that is, $W_n = \sum_{i=1}^{n-L} 1(T_{[\mathrm{i},i+L-1]} = T_{[n,n+L-1]})$. For Markov sources, the result of Jacquet and Szpankowski [23] can prove $E[W_n] = O(1)$.

W_n can be defined on the basis of the associated suffix tree S_n constructed from the first n suffixes of the data T to obtain an accurate representation of the asymptotic distribution of W_n and its factorial moments. In view of the insertion node of the $(n+1)$th suffix, when the $(n+1)$th suffix of T is inserted into S_n, W_n is equal to the number of the leaf nodes of the subtree of which the root node is the $(n+1)$th insertion node. For example, suppose the $(n+1)$th suffix begins with $\omega\beta$ for a certain $\beta \in \boldsymbol{A} = \{0, 1\}$, and $\omega \in \boldsymbol{A}*$. Then, the first n suffixes are checked. If k suffixes starting with $\omega\alpha$ ($\alpha = 1 \oplus \beta$, where $\oplus$ is the modulo 2 addition) exist and the other $(n-k)$ suffixes do not start with ω, then $W_n = k$ can be obtained; that is, W_n is the size of the subtree starting from the newly inserted branch point.

The next goal is to study W_n in the suffix tree, which is constructed from a string T generated by a binary memoryless source. However, the strings in the suffix tree are highly dependent on each other; thus, W_n is difficult to analyze accurately. Therefore, to study W_n, the parameter M_n is defined to have the same asymptotic distribution as W_n; thus, the problem is reduced to a simple asymptotic equivalence problem by analyzing the random tree constructed from n independent sequences generated by memoryless sources.

First, a mathematical expression is presented to analyze the problem. Let $0 < p < 1$, $q = 1 - p$. Define $X(j)$ as the sequence $X_1^{(j)} X_2^{(j)} X_3^{(j)} \ldots$, where $\{X_i^{(j)} | i, j \in \mathbb{N}\}$ is a set of independent and identically distributed random variables on $\{0, 1\}$, and $P\left\{X_i^{(j)} = 0\right\} = p$. Let $l_j^{(n)} = \sup\{i \geq 0 | X_1^{(j)} \ldots X_i^{(j)} = X_1^{(n+1)} \ldots X_i^{(n+1)}\}$.

Define $L_n = max_{j \leq n} l_j^{(n)}$. Search each of the first n strings for the matching prefixes which match the prefixes in the $(n+1)$th string. Then the length of the longest matching prefix is calculated and recorded as L_n. Finally, $M_n = \#\{j | 1 \leq j \leq n, l_j^{(n)} = L_n\}$ is defined; thus, M_n represents the number of the longest matching prefixes of the length L_n. Let $M_0 = 0$. If the string $X(1), \ldots, X(n)$ is a suffix of T, then M_n is asymptotically equivalent to the W_n defined above.

$C_{j_1,\ldots,j_k}$ is the length of the longest identical prefix in k strings $X(j_1), \ldots, X(j_k)$, and $l_j^{(n)} = C_{j,n+1}$. In the tree constructed by $(n + 1)$ strings, the depth of the kth string, $D_{n+1}(k)$, is the path length from the root node of the tree to the leaf node containing the kth string. Given $D_{n+1}(n + 1) = \max_{1 \leq j \leq n} C_{j,n+1} + 1$, $L_n = D_{n+1}(n + 1) - 1$ can be obtained. Therefore, $M_n = \#\{j | 1 \leq j \leq n, C_{j,n+1} + 1 = D_{n+1}(n + 1)$; that is, M_n represents the size of the subtree starting from the newly inserted branch node.

The exponential generation function is defined as

$$G(z, u) = \sum\nolimits_{n \geq 0} E[u^{M_n}] \frac{z^n}{n!}$$
$$\text{and } W_j(z) = \sum\nolimits_{n \geq 0} E[(M_n)^{\underline{j}}] \frac{z^n}{n!},$$

where the plural $u \in C$ and $j \in \mathbb{N}$. If $f : \mathbb{C} \to \mathbb{C}$, then the recursive relationship

$$\begin{aligned} E[f(M_n)] &= p^n (qf(n) + pE[f(M_n)]) + q^n (pf(n) \\ &\quad + qE[f(M_n)]) + \sum\nolimits_{k=1}^{n-1} \binom{n}{k} p^k q^{n-k} \\ &\quad \times (pE[f(M_k)] + qE[f(M_{n-k})]) \end{aligned} \tag{2}$$

holds for all $n \in \mathbb{N}$. If $f(0) = 0$, then the recursion is also true when $n = 0$. To verify the recursion, the possible value of $X_1^{(j)}$ is considered only when $1 \leq j \leq n + 1$.

First, if $n \in \mathbb{N}$, then

$$\begin{aligned} E\left[u^{M_n}\right] &= p^n \left(qu^n + pE\left[u^{M_n}\right]\right) + q^n \left(pu^n + qE\left[u^{M_n}\right]\right) \\ &\quad + \sum\nolimits_{k=1}^{n-1} \binom{n}{k} p^k q^{n-k} \left(pE\left[u^{M_k}\right] + qE\left[u^{M_{n-k}}\right]\right) \end{aligned} \tag{3}$$

If $j \in \mathbb{N}$ and $n \geq 0$, then

$$\begin{aligned} &E[(M_n)^{\underline{j}}] \\ &= p^n (qn^{\underline{j}} + pE[(M_n)^{\underline{j}}]) + q^n \left(pn^{\underline{j}} + qE\left[(M_n)^{\underline{j}}\right]\right) \\ &\quad + \sum\nolimits_{k=1}^{n-1} \binom{n}{k} p^k q^{n-k} \left(pE\left[(M_k)^{\underline{j}}\right] + qE[(M_{n-k})^{\underline{j}}]\right) \end{aligned} \tag{4}$$

The asymptotic solution of these recursive relationship can be derived by using the research results of Ward and Szpankowski [24].

Let $z_k = \frac{2kr\pi i}{\ln p}$ hold for all $k \in Z$, and $\frac{\ln p}{\ln q} = \frac{r}{s} (r, s \in Z)$. Then,

$$\begin{aligned} E\left[(M_n)^{\underline{j}}\right] &= \Gamma(j) \frac{q\left(\frac{p}{q}\right)^j + p\left(\frac{q}{p}\right)^j}{h} + \delta_j\left(log_{\frac{1}{p}} n\right) \\ &\quad - \frac{1}{2} n \left(\frac{d^2}{dz^2} \delta_j \left(log_{\frac{1}{p}} z\right)\right)\Big|_{z=n} + O\left(n^{-2}\right) \end{aligned} \tag{5}$$

where

$$\delta_j(t)=\sum\nolimits_{k\neq 0}-\frac{e^{2kr\pi it}\Gamma(z_k+j)(p^j q^{-z_k-j+1}+q^j p^{-z_k-j+1})}{p^{-z_k+1}\ln p+q^{-z_k+1}\ln q} \tag{6}$$

Γ is the Euler gamma function.

The term $-\frac{1}{2}n\left(\frac{d^2}{dz^2}\delta_j\left(log_{\frac{1}{p}}z\right)\right)\Big|_{z=n}$ in the (5) satisfies $O(n^{-1})$. δ_j is a periodic function with a small fluctuation range. If $\ln p/\ln q$ is an irrational number, then $\delta_j(x)\rightarrow 0$ when $x\rightarrow\infty$. The asymptotic distribution of M_n is described as follows.

Let $z_k=\frac{2kr\pi i}{\ln p}$ hold for all $k\in Z$, and $\frac{\ln p}{\ln q}=\frac{r}{s}(r,s\in Z)$. Then,

$$E\left[u^{M_n}\right]=-\frac{q ln(1-pu)+p ln(1-qu)}{h}+\delta\left(log_{\frac{1}{p}}n,u\right)-\frac{1}{2}n\left(\frac{d^2}{dz^2}\delta_j\left(log_{\frac{1}{p}}z,u\right)\right)\Big|_{z=n}+O(n^{-2}) \tag{7}$$

where (8), as shown at the bottom of this page. Γ is the Euler gamma function. The following equation (9) can be obtained, as shown at the bottom of this page. Then, (10), as shown at the bottom of this page. If $\ln p/\ln q$ is irrational and u is fixed, then $\delta(x,u)\rightarrow 0$ when $x\rightarrow\infty$. And $P(M_n=j)\approx\frac{p^jq+q^jp}{jh}$. Therefore, it is completely possible to use the multiplicity of matching in LZSRD compression for error correction.

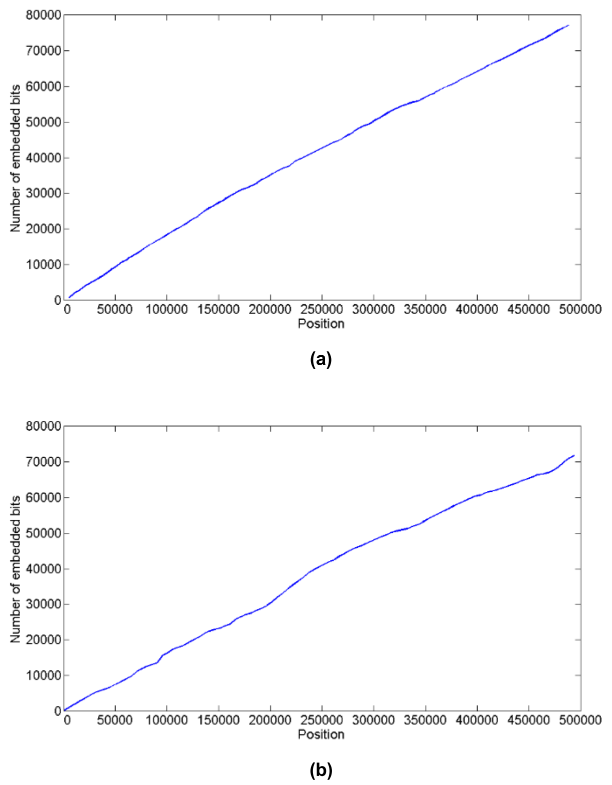

FIGURE 6. Relationship between the number of embedded bits and the file length.

V. EXPERIMENTAL AND PERFORMANCE ANALYSIS

The Canterbury Corpus [25] and the Calgary Corpus [26] are the collection of files used for benchmarking lossless data compression algorithms. The two corpora are used to conduct experiments in this study. The standard LZSS encoder uses a sliding window with a length of 32 KB, and the maximum length of the matching phrase is 256.

A. EMBEDDING CAPABILITY OF LZSS

LZSS is a sliding window dictionary coding method. At the initial stage of compression, the number of the longest matching phrases is small because there are not many entries in the dictionary. As the compression progresses, the dictionary fills up and the number of longest matching phrases gradually increases and tends to be stable. The proposed method in this study uses the check code carried in the redundant to protect the data. The embedding capability of the LZSS compressed data is tested at first. In the experiment we measure the $\log_2(r)$. Figures 6(a) and 6(b) show, respectively, the average of the number of the bits that can be embedded in the compressed data when the files in the Canterbury Corpus and the Calgary Corpus are compressed into LZSS files. Figure 6 clearly shows that the number of bits embedded in the file F increases linearly with the length $|F|$ of the file. The packet data with the length of 255-2e requires 2e parity bits to correct e errors. Correcting e errors is possible as long as the embedding rate of the compressed data is not less than 2e/(255-2e). The experimental results show that when the number of the longest matching phrases in compressing data by LZSS is stabilized, the embedding rate can be satisfied in each data packet, when $e=2$ or even bigger than 2.

$$\delta(t,u)=\sum\nolimits_{k\neq 0}-\frac{e^{2kr\pi it}\Gamma(z_k)\left(q(1-pu)^{-z_k}+p(1-qu)^{-z_k}-p^{-z_k+1}-q^{-z_k+1}\right)}{p^{-z_k+1}\ln p+q^{-z_k+1}\ln q} \tag{8}$$

$$E\left[u^{M_n}\right]=\sum\nolimits_{j=1}^{\infty}\left[\frac{p^jq+q^jp}{jh}+\sum\nolimits_{k\neq 0}-\frac{e^{2kr\pi it log_{\frac{1}{p}}n}\Gamma(z_k)\left(p^jq+q^jp\right)(z_k)^{\bar{j}}}{j!\left(p^{-z_k+1}\ln p+q^{-z_k+1}\ln q\right)}\right]u^j+O(n^{-1}) \tag{9}$$

$$P(M_n=j)=\frac{p^jq+q^jp}{jh}+\sum\nolimits_{k\neq 0}-\frac{e^{2kr\pi it log_{\frac{1}{p}}n}\Gamma(z_k)\left(p^jq+q^jp\right)(z_k)^{\bar{j}}}{j!\left(p^{-z_k+1}\ln p+q^{-z_k+1}\ln q\right)}+O\left(n^{-1}\right) \tag{10}$$

B. ERROR REPAIR CAPABILITY OF LZSRD

In this section, the error repair capability of LZSRD is compared with that of the method proposed in [8]. The method proposed in [8] utilizes compression coding and grammar rules to determine the error position and correct the errors. The present work uses the residual redundancy left by the LZSS encoder to carry the verification information to find and repair errors.

To test the error repair capability, all files in the Canterbury Corpus and the Calgary Corpus are compressed into LZSS files, and errors of different quantity are introduced to and randomly distributed in the LZSS compressed files. Every time the quantity of the injected errors (represented by bit error rate (BER)) changes, 100 experiments are performed on each compressed file. In the experiments, LZSRD and the method proposed in [8] are used to repair the errors and decompress the files. The average probability of successful error correction and decompression of the files is calculated and recorded.

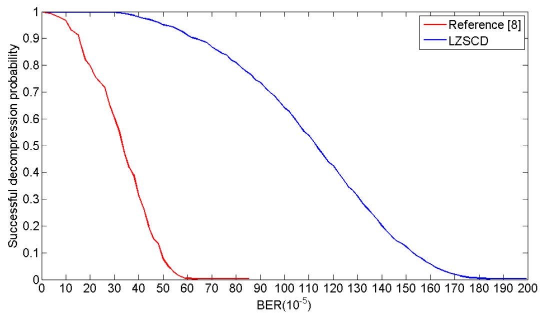

FIGURE 7. Comparison of error repair capabilities.

A comparison of the experimental results of the two methods is shown in Fig. 7. The blue and red curves represent the average probability of the successful decompression of LZSRD and the method proposed in [8], respectively. As shown in Fig. 7, when $BER = 6 \times 10^{-4}$, the method proposed in [8] can hardly decompress the file correctly, and the probability that LZSRD can successfully decompress is approximately 0.9. Therefore, the error repair capability of LZSRD is considerably higher than that of the method proposed in [8] because the latter cannot find a replacing error that does not destroy the dictionary structure; moreover, the error bit may be corrected only when the interval between the error bits is greater than the error detection delay. As BER increases, the error bits cannot be corrected once the interval is less than the error detection delay. Therefore, the error correction capability of the method proposed in [8] is low. The proposed method in the present study can repair the errors and successfully decompress data as long as the number of errors in each packet data does not exceed the verification capability.

C. PRACTICALITY OF LZSRD

All files in the Canterbury Corpus and the Calgary Corpus are compressed into the LZSS files. Some uniformly distributed error bits are added to each LZSS file to generate a corrupted LZSS file. When $BER = 10^{-5}$, LZSRD and the method proposed in [8] are used to correct the errors and decompress the file. The average probability of successful error correction and decompression of the files is calculated and recorded.

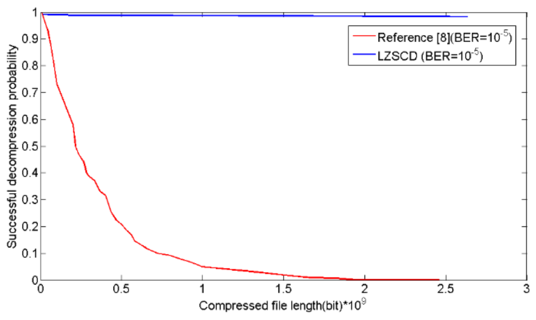

FIGURE 8. Relationship between error correction rate and length of compressed file.

The experimental results are shown in Fig. 8. The blue and red curves represent the average probability of the successful decompression of LZSRD and the method proposed in [8], respectively. As shown in Fig. 8, as the length of the compressed file increases, the error correction capability of the method proposed in [8] decreases exponentially. This result is caused by the fact that if the position of the error bit in the LZSS file is random, then the error repair rate of the method proposed in [8] is less than 1. This method will eventually fail when the length of the LZSS file is infinite because its error correcting mode can only correct up to 1 bit at a time. The RS code used in this study has stable error repair capability, and the error correction rate is only affected by the BER and thus hardly affected by the file length.

VI. CONCLUSION

LZSS is a widely used lossless data compression method in the field of general compression. However, the propagation of errors in the decompression process limits its application. To effectively repair the errors of LZSS compressed data, the existing methods need add extra information bits to protect data. These methods decrease the compression performance and are incompatible with standard algorithms. To address these shortcomings, this study proposes a new scheme compatible with the standard LZSS algorithm. It does not change the encoding rules and data format, so the data compressed by the new method can still be decompressed by the standard LZSS decoder. The new scheme does not require any additional bits. It has great error repair capability and does not affect the compression performance.

REFERENCES

[1] M. Drmota and W. Szpankowski, "Redundancy of lossless data compression for known sources by analytic methods," *Found. Trends Commun. Inf. Theory*, vol. 13, no. 4, pp. 277–417, May 2016.
[2] S. Verdú and I. Kontoyiannis, "Optimal lossless data compression: Non-asymptotics and asymptotics," *IEEE Trans. Inf. Theory*, vol. 60, no. 2, pp. 777–795, Feb. 2014.

[3] 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, Dec. 2016.
[4] F. Jiang *et al.*, "Compressed vision information restoration based on cloud prior and local prior," *IEEE Access*, vol. 2, pp. 1117–1127, Aug. 2014.
[5] G. D. Menghwar and C. F. Mecklenbrauker, "Cooperative versus non-cooperative communications," in *Proc. 2nd Int. Conf. Comput., Control Commun.*, Karachi, Pakistan, Feb. 2009, pp. 1–3.
[6] B. M. Hamschin, J. D. Ferguson, and M. T. Grabbe, "Interception of multiple low-power linear frequency modulated continuous wave signals," *IEEE Trans. Aerosp. Electron. Syst.*, vol. 53, no. 2, pp. 789–804, Apr. 2017.
[7] B. Kwon, M. Gong, and S. Lee, "Novel error detection algorithm for LZSS compressed data," *IEEE Access*, vol. 5, pp. 8940–8947, May 2017.
[8] D. Wang, X. Zhao, and Q. Sun, "Novel fault-tolerant decompression method of corrupted Huffman files," *Wireless Pers. Commun.*, vol. 102, no. 4, pp. 2499–2518, Oct. 2018.
[9] V. Kostina, Y. Polyanskiy, and S. Verdú, "Variable-length compression allowing errors," *IEEE Trans. Inf. Theory*, vol. 61, no. 8, pp. 4316–4330, Aug. 2015.
[10] J. Zhang, E.-H. Yang, and J. C. Kieffer, "A universal grammar-based code for lossless compression of binary trees," *IEEE Trans. Inf. Theory*, vol. 60, no. 3, pp. 1373–1386, Mar. 2014.
[11] S. T. Klein and D. Shapira, "Practical fixed length Lempel–Ziv coding," *Discrete Appl. Math.*, vol. 163, pp. 326–333, Jan. 2014.
[12] 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.
[13] Z. C. Pereira, M. E. Pellenz, R. D. Souza, and M. A. A. Siqueira, "Unequal error protection for LZSS compressed data using Reed-Solomon codes," *IET Commun.*, vol. 1, no. 4, pp. 612–617, Aug. 2007.
[14] G. Lakhani, "Reducing coding redundancy in LZW," *Inf. Sci.*, vol. 176, no. 10, pp. 1417–1434, May 2006.
[15] B. Park, A. Savoldi, P. Gubian, J. Park, S. H. Lee, and S. Lee, "Recovery of damaged compressed files for digital forensic purposes," in *Proc. Int. Conf. Multimedia Ubiquitous Eng. (MUE)*, Busan, South Korea, Apr. 2008, pp. 365–372.
[16] Y. Murin, R. Dabora, and D. Gündüz, "On joint source-channel coding for correlated sources over multiple-access relay channels," *IEEE Trans. Inf. Theory*, vol. 60, no. 10, pp. 6231–6253, Oct. 2014.
[17] D. Kempa and D. Kosolobov, "LZ-end parsing in compressed space," in *Proc. Data Compress. Conf. (DCC)*, Snowbird, UT, USA, Apr. 2017, pp. 350–359.
[18] G. Louchard and W. Szpankowski, "On the average redundancy rate of the Lempel-Ziv code," *IEEE Trans. Inf. Theory*, vol. 43, no. 1, pp. 2–8, Jan. 1997.
[19] S. Das, D. M. Bull, and P. N. Whatmough, "Error-resilient design techniques for reliable and dependable computing," *IEEE Trans. Device Mater. Rel.*, vol. 15, no. 1, pp. 24–34, Mar. 2015.
[20] H. H. Do, J. Jansson, K. Sadakane, and W.-K. Sung, "Fast relative Lempel–Ziv self-index for similar sequences," *Theor. Comput. Sci.*, vol. 532, no. 1, pp. 14–30, May 2014.
[21] M. J. Atallah and S. Lonardi, "Augmenting LZ-77 with authentication and integrity assurance capabilities," *Concurrency Comput., Pract. Exper.*, vol. 16, no. 11, pp. 1063–1076, Sep. 2004.
[22] I. S. Reed and G. Solomon, "Polynomial codes over certain finite fields," *J. Soc. Ind. Appl. Math.*, vol. 8, no. 2, pp. 300–304, Jun. 1960.
[23] P. Jacquet and W. Szpankowski, "Average size of a suffix tree for Markov sources," in *Proc. 27th Int. Conf. Probabilistic, Combinat. Asymptotic Methods Anal. Algorithms*, Krakow, Poland, 2016, pp. 1–13.
[24] M. D. Ward and W. Szpankowski, "Analysis of a randomized selection algorithm motivated by the LZ'77 scheme," in *Proc. 6th Workshop Algorithm Eng. Exp. 1st Workshop Anal. Algorithmics Combinat.*, New Orleans, LA, USA, 2004, pp. 153–160.
[25] *The Calgary Corpus*. Accessed: Mar. 30, 2018. [Online]. Available: http://corpus.canterbury.ac.nz/descriptions/#cantrbry
[26] *The Calgary Corpus*. Accessed: Mar. 30, 2018. [Online]. Available: http://corpus.canterbury.ac.nz/descriptions/#calgary

GANG WANG received the B.S. degree in signal analysis and the M.S. degree in information and communication engineering from the National Digital Switching System Engineering and Technology Research Center of China, Zhengzhou, China. He is currently pursuing the Ph.D. degree in information engineering.

He has been with the National Digital Switching System Engineering and Technology Research Center. His research interests include signal analysis, information processing, and pattern recognition.

HUA PENG is currently a Professor and a Ph.D. Supervisor with the National Digital Switching System Engineering and Technology Research Center of China, Zhengzhou, China. He is also the Head of the Department with the Signal Analysis and Processing Laboratory. His research interests include communication signal processing and software-defined radio.

YONGWANG TANG received the Ph.D. degree in control theory and control engineering from the Kunming University of Science and Technology, Kunming, China. He is currently an Associate Professor. His research interests include intelligent information processing, semantic web technologies, information fusion, and database management.