Sensors and Materials, Vol. 36, No. 12 (2024) 5233–5246
MYU Tokyo

S & M 3864

Lossless Data Compression of Wireless Sensor in Bridge Inspection System

Zhengsong Ni,[1] Shuri Cai,[2*] and Cairong Ni[2]

[1]College of Big Data and Artificial Intelligence, Fujian Polytechnic Normal University, No. 1 Campus New Village, Longjiang Street, Fuqing, Fuzhou, Fujian 350300, China
[2]Institute of Highway Science, Ministry of Transport, Beijing University of Posts and Telecommunications, 8 Xitucheng Road, Haidian District, Beijing 100086, China

(Received April 30, 2024; accepted November 29, 2024)

Keywords: lossless data compression algorithm, wireless sensor network, bridge inspection, Huffman compression algorithm, dictionary model

In bridge inspection using wireless technology, the sensor node carries very limited energy. In this paper, data compression technology to reduce the amount of communication data by sacrificing data accuracy is introduced. The lossless data compression algorithm was chosen as the algorithm of the bridge inspection system, and the algorithm was further improved. In addition, we compared the compression rates and execution times of the Huffman, Lempel–Ziv–Storer–Szymanski (LZSS), and improved Lempel–Ziv–Welch (LZW) algorithms. The results showed that the improved LZW algorithm is highly adaptable, accommodates any type of input data, and can make full use of the repeatability of data. Therefore, this lossless data compression algorithm is suitable for application in the wireless sensor networks of bridge inspection systems.

1. Introduction

In recent years, the health status monitoring of bridges has attracted increasing attention, because the safety and reliability of bridges are crucial to ensure the smooth progress of transportation. Wireless sensor networks (WSNs) are widely used in bridge health monitoring systems, and a large number of sensor nodes are deployed to collect the structural information of bridges in real time.(1) However, owing to the resource constraints of sensor nodes, such as computing power, storage capacity, and energy supply, the data compression of sensor nodes becomes particularly important and necessary.

The lossless data compression algorithm is an effective data compression method that can reduce the amount of data storage and transmission by deleting or retaining redundant information without losing the original data information.(2) In the bridge inspection system, sensor nodes usually collect a large amount of data, such as vibration, temperature, and displacement, which contain the key information of the running state and health status of the bridge structure. Therefore, it is very important to find a balance between maintaining data integrity and reducing the amount of data transferred.

*Corresponding author: e-mail: caishuri@126.com
https://doi.org/10.18494/SAM5120

ISSN 0914-4935

The purpose of this work is to study the application of the lossless data compression algorithm in WSNs to bridge inspection systems.(3) First, we will analyze the data compression algorithms commonly used today and evaluate their applicability to bridge inspection systems. Then, we will propose a bridge inspection system architecture based on the lossless data compression algorithm and describe its implementation and key technologies in detail. Next, we will devise a series of experiments to validate the performance and efficacy of this algorithm, and contrast its results with those of alternative compression algorithms. Finally, on the basis of experimental results, we will discuss the advantages and limitations of the lossless data compression algorithm in bridge inspection systems and put forward the improvement direction and future research prospects.(4)

The objective of this research is to reduce the data transmission of sensor nodes using the lossless data compression algorithm, so as to extend the energy life of sensor nodes and improve the reliability and stability of the system.(5) In addition, the lossless data compression algorithm can reduce the data storage capacity and data transmission delay and improve the real-time performance of the system. Through the real-time monitoring and analysis of the bridge structure, we can find the potential structural problems in time and take the corresponding measures to ensure the safe operation of the bridge.

The results of this study are of great significance to the development of bridge engineering and WSN technology. By applying the lossless data compression algorithm to the bridge inspection system, we can improve the energy utilization efficiency of sensor nodes, reduce the maintenance cost of the system, and promote the application of WSN technology in the field of bridge health monitoring.(6) In addition, this research can also provide reference for other fields of data compression and transmission problems and has a wide range of application prospects.

Therefore, the purposes of this study are to explore an efficient and reliable bridge health monitoring method through the study of the lossless data compression algorithm on bridge inspection systems and to provide theoretical and empirical support for research and practice in related fields. Through this research, we hope to improve the safety and reliability of bridges and contribute to social transportation construction and economic development.

2. Research Status of Lossless Data Compression Algorithms at Home and Abroad

WSNs play an important role in bridge inspection systems, which can monitor and collect the data of bridge structure in real time to provide efficient detection and early warning. The research of lossless data compression algorithms in this field is aimed at reducing the storage and transmission overhead of data while maintaining the accuracy and integrity of data. The following is a brief overview of the research status of the lossless data compression algorithm in bridge inspection systems at home and abroad.(7)

2.1 Domestic research status

(1) Compression algorithm based on wavelet transform: Wavelet transform is used to analyze and compress bridge data in the frequency domain to reduce data redundancy. For example,

algorithms using wavelet transform and adaptive threshold selection can achieve efficient compression and recovery.[8]

(2) Adaptive dictionary learning algorithm: This algorithm can learn and generate dictionaries suitable for data representation in accordance with the characteristics of bridge data to reduce the representation error of data. By optimizing the dictionary learning process, better data compression results can be achieved.[9]

(3) Sparse representation algorithm: The sparse representation algorithm is based on the assumption that bridge data can be represented by fewer underlying signals. Through sparse representation, data can be compressed and recovered efficiently.[10]

2.2 Foreign research status

(1) Compressed sensing algorithm: Compressed sensing is an emerging signal processing technology that can restore the original signal through a small amount of sampling. In bridge inspection systems, the compressed sensing algorithm can effectively reduce the data storage and transmission overhead.[11]

(2) Non-negative matrix decomposition algorithm: The non-negative matrix decomposition algorithm can decompose and compress the bridge data and maintain the non-negative data. This algorithm achieves good compression results in the damage detection and diagnosis of bridge structures.[12]

(3) Adaptive compression algorithm: This algorithm adaptively selects compression methods and parameters in accordance with the characteristics of bridge data and adjusts the compression ratio in real time. By dynamically adjusting the compression ratio, efficient compression can be achieved while ensuring data accuracy.[13]

In general, the research of the lossless data compression algorithm in bridge inspection systems has advanced considerably. Various algorithms and methods have been proposed by researchers at home and abroad, and some positive results have been obtained. However, there is still room for further research and improvement to improve the effectiveness and performance of the compression algorithm and make it better applicable to bridge inspection systems.

3. Classification of Data Compression Technology

There are many ways to classify data compression. According to some statistics, it can reach 20 to 40 kinds, which have not yet been unified. Most scholars agree that data compression is divided into lossless compression and lossy compression.

Data compression can be classified on the basis of different classification criteria. The following are common data compression categories:

(1) Lossy Compression: Lossy compression is a compression method that compresses data while losing some of the details and precision of the original data. This compression method is suitable for situations where data accuracy is relatively low, such as audio compression and video compression.[14]

(2) Lossless Compression: Lossless compression is a compression method that can completely retain the accuracy and integrity of the original data during compression and decompression.

This compression method is suitable for situations where data accuracy is required, such as text and image compression.[15]

(3) Dictionary-based Compression: Dictionary-based compression is a lossless compression method that compresses data by creating and using dictionaries. Common dictionary compression algorithms include the Lempel–Ziv–Welch (LZW) algorithm and Huffman coding.[16]

(4) Predictive Coding: Predictive coding is a lossless compression method that utilizes the statistical properties of data and predictive models to achieve data compression. Common predictive coding algorithms include differential coding and arithmetic coding.

(5) Transform Coding: Transform coding is a lossless compression method that reduces the redundancy of data by transforming the data. The most common conversion coding algorithms are JPEG image compression based on discrete cosine transform (DCT) and JPEG2000 image compression based on discrete wavelet transform (DWT).[17]

(6) Entropy Coding: Entropy coding is a lossless compression method that uses the concept of information entropy to represent the probability of symbol occurrence. The symbols that appear with high frequency are represented by short code words, and the symbols that appear with low frequency are represented by long code words, so as to achieve data compression. Common entropy coding algorithms include Huffman coding and arithmetic coding.[18]

3.1 Several common traditional lossless compression algorithms

The above classifications are not mutually exclusive, and some compression algorithms can use different compression techniques simultaneously. The choice of the appropriate compression method depends on the characteristics of the data, the application requirements, and the performance requirements of compression and decompression.

There are many traditional lossless compression algorithms, and the following are some common ones.

(1) Huffman Coding: Huffman coding is an entropy coding method that uses variable-length code words to represent the occurrence probability of different symbols. Symbols that appear with high frequency are represented by short code words, and symbols that appear with low frequency are represented by long code words. This encoding method is often used in text compression and image compression.

(2) Arithmetic Coding: Arithmetic coding is also an entropy coding method that maps the entire message sequence into an interval and scales the interval size in accordance with the probability distribution of the message. Arisso coding can represent the probability of the occurrence of symbols more precisely, so it can achieve a higher compression efficiency than Hoffmann coding.

(3) LZ77 and LZ78 algorithms: LZ77 and LZ78 algorithms are a class of dictionary-based compression algorithms. The LZ77 algorithm finds the longest matching string by using a sliding window and represents it with a pointer and length. The LZ78 algorithm uses a dictionary to store the strings that have been encountered and represents them with pointers

and new letters. These algorithms have achieved good compression effects in practical application.

(4) Run-length Encoding (RLE): RLE is a simple compression method that compresses data by counting the number of consecutive occurrences of the same symbol. Repeated symbols need only be stored once and then represented by degrees. This encoding method is very effective when dealing with continuously recurring symbols, such as white space in an image. These traditional lossless compression algorithms have been widely used in many applications and have achieved good compression results. However, with the increase in data volume and the change of application requirements, new compression algorithms are also emerging to better meet the compression needs of different types of data.

3.2. Comparison of several lossless compression algorithms

There are several key factors to consider when comparing different lossless compression algorithms.

(1) Compression rate: The compression rate is an important indicator used to measure the compression effect of the algorithm and represents the ratio of the compressed data size to the original data size. The ideal algorithm should be able to provide a higher compression rate, i.e., a smaller compressed file size.

(2) Compression speed: Compression speed refers to the time taken by the compression algorithm to perform the compression operation. For large data sets or scenarios that require real-time compression, high compression speeds are an important consideration.

(3) Decompression speed: The decompression speed refers to the time taken by the compression algorithm to perform the decompression operation. High decompression speeds are important in scenarios where data need to be decompressed frequently.

(4) Compression quality: Compression quality indicates the degree of difference between the compressed data and the original data. A higher compression quality means that the compressed data can be closer to the original data without losing key information.

(5) Applicable data types: Different compression algorithms have different effects on different types of data. For example, some algorithms perform well when working with text data and may be less effective when working with images or audio data. Therefore, it is necessary to consider the data type and application scenario to choose the appropriate compression algorithm.

In general, there is no single algorithm that is the best at everything. Choosing the right compression algorithm depends on your specific needs and limitations. In practical applications, factors such as compression rate, compression speed, and decompression speed are usually weighed to choose the most suitable algorithm. At the same time, you can also consider using a combination of multiple algorithms or using adaptive algorithms to obtain the best compression effect for different data types and scenarios.

4. Research and Implementation of LZW Improvement Algorithm

4.1 Principle of LZW algorithm

The LZW algorithm works as follows.

(1) Initialize the dictionary: Create an initial dictionary that contains all possible single characters as keys, corresponding to their index in the dictionary.
(2) Read input: Read a character from the input data as the current input character.
(3) Check the dictionary: Concatenate the current input character and the previously read character sequence together to form a new string. Check whether the string exists in the dictionary.
a. If it exists, continue reading the next input character, concatenating the current string with the previously read character sequence.
b. If it does not exist, output the encoding of the previously read character sequence, add the current string to the dictionary, and assign it a new encoding.
(4) Output encoding: Output the encoding of the previously read character sequence.
(5) Update the dictionary: Add the current string to the dictionary and assign it a new encoding.
(6) Repeat Step 2 until the data input is complete.

The core idea of the LZW algorithm is to realize compression by building the dictionary step by step. It uses the repeated occurrence of a string to reduce the redundancy of data, and the repeated string is represented by the index value, so as to achieve the compression effect of data. When decompressing, the original data can be recovered by remapping the encoding of the output to a string in the dictionary.

Note that the LZW algorithm is a dynamic dictionary compression algorithm, which will continuously increase the size of the dictionary during compression. This makes the LZW algorithm very effective when dealing with data with a large number of repeating strings, such as text data. The LZW algorithm is an improvement on the LZ78 algorithm similar to the LZSS improvement on the LZ77 algorithm, that is, the LZW compressor does not encode a single character. The compression principle is that by analyzing the input data stream, a string table is generated adaptively while encoding, and all the strings that have appeared before are not repeated. The compression process is as follows. By comparing the current input data stream with the strings in the string table, the output value is determined. Simultaneously, the string table is updated. This is illustrated in Fig. 1.

In addition, to improve the storage space overhead caused by reserving 256 descendants for each node in LZ78, the improved LZW algorithm uses a hash function to identify and find the child nodes of a given node. The specific method is to use parent_code and child_code to obtain the offset of a child node. When you find the corresponding child node on the basis of this offset, check whether this node is used by other nodes to ensure that no conflict occurs. A typical compressed reality, like that in Table 1, has an input data stream of /WED/WE/WEE/WEB/WET.

The algorithm preloads a single symbol into the dictionary's data structure. In this way, even if the symbol appears in the input data stream for the first time, there is no phenomenon that cannot be encoded immediately. During the implementation of the algorithm, LZW tries to

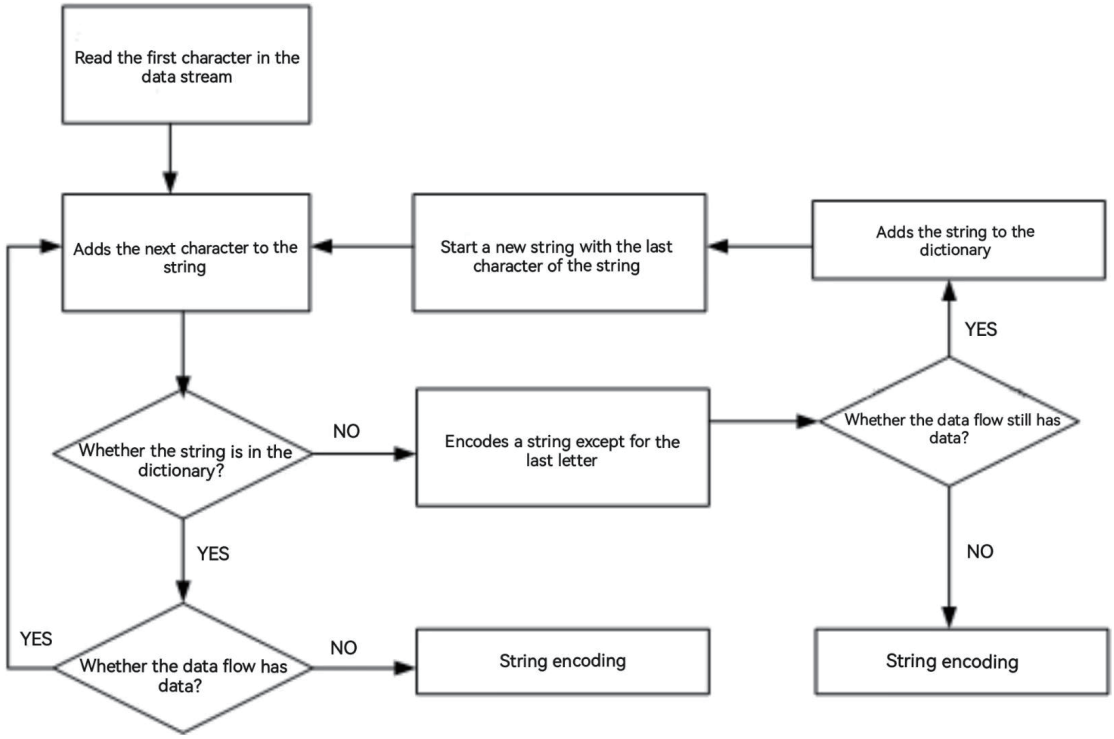

Fig. 1. LZW algorithm compression process.

Table 1
LZW algorithm compression instance.

Input string = /WED/WE/WEE/WEB/WET 0.9991 0.9562 0.9840			
Character input	Code output	New code value	New string
/W	/	256	/W
E	W	257	WE
D	E	258	ED
/	D	259	D/
WE	256	260	/WE
/	E	261	E/
WEE	260	262	/WEE
/W	261	263	E/W
EB	257	264	WEB
/	B	265	B/
WET	260	266	/WET
EOF	T		

output code for symbols in the data stream, and when there is a new code output, the corresponding new phrase is added to the dictionary. As shown in Fig. 2, the improved LZW algorithm uses the hash function to determine and find the children of a given node. The specific method is to use the parent and child nodes to obtain the offset of a descendant node. When the corresponding descendant node is found using this offset, it should be checked whether this node is used by other nodes to ensure that there is no conflict (see Table 2).

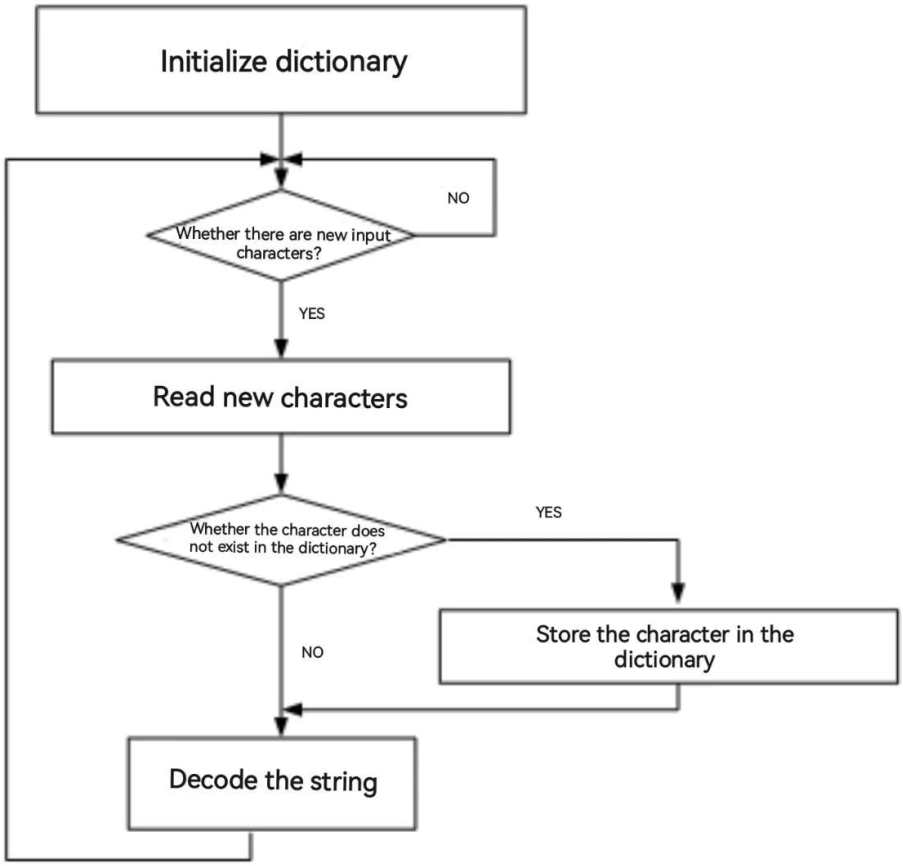

Fig. 2. Improved LZW algorithm flow chart.

Table 2
Improved LZW algorithm compression.

Input Codes: / W E D 256 E 260 261 257 B 260 T				
Input/ NEW_CODE	OLD_CODE	STRING/ Output	CHARACTER	New table entry
/	/	/		
W	/	W	W	256 = /W
E	W	E	E	257 = WE
D	E	D	D	258 = ED
256	D	/W	/	259 = D/
E	256	E	E	260 = /WE
260	E	/WE	/	261 = E/
261	260	E/	E	262 = /WEE
257	261	WE	W	263 = E/W
B	257	B	B	264 = WEB
260	B	/WE	/	265 = B/
T	260	T	T	266 = /WET

The maintenance of the dictionary is relatively simple during the decompression of the improved LZW algorithm. Instead of searching down the tree, the code is read directly from the encoded stream and then searched up the tree. As long as the parent node is correctly defined in

the dictionary, it can be correctly decoded. Therefore, the tree built in the decompression algorithm does not need to use a hash function and does not need to find an index.

One problem with the search up tree is that the decoded characters are in reverse order, so they have to be pushed onto the stack, ejected in reverse order, and written to the output file.

The improved LZW decompression algorithm is as follows.

```
read OLD _ CODE
output OLD _ CODE
   WHILE there are still input characters DO
      Read NEW _ CODE
      STRING=get translation of NEW _ CODE
      output STRING
   CHARACTER = first character in STRING
   Add OLD _ CODE+CHARACTER to the translation table
   OLD _ CODE=NEW _ CODE
   END of WHILE
```

The improved LZW algorithm has no transmission overhead and is simple to calculate. Both the sender and the receiver contain initializing dictionaries to create new dictionary entries on the basis of existing dictionary entries. As soon as the data is received, the receiver can recreate the dictionary. Its biggest features are simple logic, easy to implement, and high speed.

4.2 Improved LZW algorithm implementation

The C program of the improved LZW algorithm is introduced below.

(1) Symbol definition

BITS: dictionary address space (0–2BITS-l), BITS={12,13,14};

HASHING_SHIFT: indicates the offset of the child node. HASHING_SHIFT=BITS-8;

Code-value []: indicates the actual code value of the node, and the output code value after the string is encoded in the compression program;

Prefix-code []: indicates the prefix node code. Each string in the dictionary has a prefix node shorter than its own.

append_character[]: current node code;

decode_stack[]: decoder stack;

(2) Subfunction definition

(a) encoding input function lzw_input_code(unsigned char*&input)

The function is employed within the LZW decompression algorithm, primarily to accomplish the transformation of encoded data formats, specifically converting 8-bit input data into BITS format output. The implementation is to first read 8 bits of data into the 32-bit length of the FIFO buffer; if the hash table address space is BITS, then output the high bits of data. The whole implementation process is like a 32-bit-long sliding window, each time taking the highest BITS of data in the window as output.

Input parameters: input is a pointer to input data, input in bytes; Returned value: BITS of data

(b) output coding function lzw_output_code(unsigned char*&output, unsigned int code)

Function Description: Used in the LZW compression program, the encoded data is output in 8-bit format, which is a function often used in the data compression program. Define a 32-bit FIFO buffer where the program runs once and outputs data in bytes until the data in the buffer is less than one byte long.

Input parameter: code indicates the current code. Returned value: output is the output data pointer

(c) Hash function dictionary address index lzw_find_match(int hash_prefix, unsigned int Hash_character)

Function description: According to the set hash (hash) function H(key) and the method of handling conflicts, the keyword is mapped to a finite continuous address set (interval), and the "image" of the keyword in the address set is taken as a storage location in the table, which is a hash table. This mapping process is called the hash table or hash, and the resulting storage location is called the hash address. The hash function applied to this algorithm is index = (hash_character<<HASHING_SHIFT) and yyyy hash_prefix; index is the address index obtained using the prefix and current nodes. If the address index value does not exist (code_value[index] is empty) or the string corresponding to code_value[index] and the prefix node hash_prefix are the same as the string composed of the current node hash_character, the value is returned. If the index exists but the string corresponding to code_value[index] and the prefix node hash_prefix are not exactly the same as the string composed of the current node hash_character, subtract an offset value from the index value and check until the index value does not exist.

hash_prefix indicates the prefix code. hash_character indicates the current character. Return value: address index.

d, decode string function *lzw_decode_string(unsigned char*buffer, unsigned int code)

Function Description: Reverse addressing is a technique that is utilized to create a stack for storing strings. The particular program that uses this method is implemented in the decompression process. If the input code is an encoded value, press the corresponding current character append_character[code] into the stack, and then look for its prefix string encoded prefix_code[code] until

Prefix_code[code] is a single character (ASCII value less than 255).

Input parameters: buffer Stack Buffer address, code is the current code; Return value: stack bottom address

(3) LZW data compression function

Define the prefix character as string_code and the input character as character. The LZW data compression algorithm flows as follows.

Step 1: Initialize the dictionary, code_value[i]=UNUSED;

Step 2: Enter the character string_code;

Step 3: Enter character;

Step 4: Check if the current position is at the end of the file. If it is, proceed to step 8. If not, proceed to step 5.

Step 5: Use the lzw_find_match() subfunction to determine the current string dictionary index;

Step 6: If the current string does not exist in a certain context (presumably a dictionary or some other storage mechanism), output the current code string_code. Then, add the current string to the dictionary and set string_code equal to a character. After that, return to Step 3. If the current string exists, go to Step 7.

Step 7: Set string_code = code_value[index] and skip back to Step 3.

Step 8: Output the last code, string_code, and mark the end of the file.

(4) LZW data decompression function

Set the old encoding to old_code, the new encoding to new_code, the top character of the stack to character, and the data output pointer to destoutPtr. The LZW decompression algorithm flow is as follows.

Step 1: Initialize;

Step 2: Use the lzw_input_code() function to read the code value old_code, character=old_code;

Step 3: Assign the old_code value to the data pointer destoutPtr. The pointer points to the next address.

Step 4: Use the lzw_input_code() function to read the code value new_code, if new_code is the end of the file, exit; otherwise, enter step 5;

Step 5: If new_code is not in the dictionary, press character: to the bottom of the stack, use lzw_code_string(*buffer, old_code) to reverse construct the string, and point the data pointer destoutPtr to the string to achieve data output; Otherwise, lzw_decode_string(*buffer, new_code) is used to reversely construct the string, and the data pointer destoutPtr points to the string to achieve data output.

Step 6: If the dictionary is not full, add the string old_code+character: to the dictionary;

Step 7: Assign the new_code value to old_code and go to Step 4.

The above is an implementation of the LZW modified algorithm for fixed-length dictionaries, where the size of the dictionary is variable, and a larger dictionary means that more and longer phrases can be stored, resulting in higher compression. The LZW improvement algorithm usually uses the 15-bit length code to have a 32K capacity phrase dictionary, but increasing the size of the dictionary brings about new problems. If the compressed file is small, using more bits of code slows down compression. For small files, the number of phrases generated in the compression may not be enough to fill the entire large-capacity dictionary, and it is more advantageous to determine the output code bits adaptively in accordance with the size of the dictionary. In response to this problem, it has been proposed that during the compression, the output of the program is a variable-length code, starting with 9 bits, until the dictionary has grown to 256 new phrases, starting with the 10- bit code, and so on until the 15-bit code is used.

4.3 Comparison of lossless data algorithms

For Huffman coding—Theoretical basis: It is based on the frequency of symbols, creating variable-length codes. Improvements: Adaptive Huffman coding adjusts the code lengths as more data are encountered, optimizing compression efficiency.

Regarding LZSS (Lempel–Ziv–Storer–Szymanski)—Theory: it uses a sliding window for finding repeated sequences and encoding them with pointers. Enhancements: Various modifications for optimizing the size of the search buffer and dictionary management for higher performance.

For LZW (Lempel–Ziv–Welch)—Principle: It constructs a prefix code as it encounters new patterns in the data stream. Advancements: Modified versions, like LZ78 and GIF's variant of LZW, improve compression speed or adapt the method for specific data types. To delve deeper, actual implementation tests could involve compressing and decompressing diverse datasets to quantify these differences under real-world conditions. These tests would help illustrate where

each algorithm shines and where it might fall short, given different data characteristics and use-case requirements.

In this section, in accordance with the above algorithm principle, the actual data collected by the sensor network are compressed by the above algorithm for further analysis.

$$\text{Compressibility definition}: \eta = \frac{Output\ file\ size\ after\ compression}{Enter\ file\ size\ before\ compression} * 100\% \quad (1)$$

The algorithm compression rates and execution times are shown in Table 3.

The Huffman algorithm incorporates a rapid coding mechanism. LZSSI, on the other hand, is a cyclic queue LZSS compression algorithm that operates on the basis of a sliding window principle. Further evolution in LZSS methodology led to LZSS2, which utilizes a binary-search-tree structure for its compression process. Its configuration includes a search buffer sized at 4096 bytes, a forward buffer extent of 17 bytes for optimization. In the case of the improved

Table 3
Algorithm compression rates and execution times.

Filename	File size (Byte)	LZW		LZSS1		LZSS2		Huffman	
		Algorithm compression ratio (%)	Execution time (ms)	Algorithm compression ratio (%)	Execution time (ms)	Algorithm compression ratio (%)	Execution time (ms)	Algorithm compression ratio (%)	Execution time (ms)
sensor012 #20071103 #093022	117789	45.0	3	24.3	168	24.5	169	32.7	7
sensor012 #20071103 #094618	600048	45.6	15	23.6	850	24.6	886	33.3	37
sensor012 #20071103 #095945	605876	45.7	15	24.3	872	24.2	869	32.9	38
sensor012 #20071103 #101623	1150048	45.5	29	23.6	1616	24.8	1698	33.1	71
sensor012 #20071103 #103228	53012	45.2	1	24.3	76	24.3	76	32.8	3
sensor012 #20071103 #104219	1198048	45.5	31	24.3	1690	24.7	1718	32.6	75
sensor012 #20071103 #105649	1200128	45.5	31	23.8	1707	24.5	1757	33.0	74
sensor012 #20071103 #111038	1205876	45.3	31	23.8	1743	24.1	1765	33.0	75
sensor012 #20081103 #112654	1390976	45.8	35	23.6	1986	24.4	2054	32.7	87
sensor012 #20071103 #113936	607768	45.7	16	23.6	853	24.8	897	33.3	37

LZW algorithm, the dictionary space has been expanded to accommodate 4096 entries to enhance compression efficiency. The following conclusions can be drawn from Table 3.

(a) The LZSS1 algorithm has the best overall compression effect, but it takes the longest time, so it is not an ideal choice for real-time transmission systems.
(b) The compression rate of the LZSS1 algorithm is similar to that of LZSS2, but the execution time of the LZSS2 algorithm is much shorter.
(c) The improved LZW algorithm has a poor compression effect compared with LZSS. This statement involves the comparison of the execution time of the improved LZW algorithm with that of the Huffman algorithm. It indicates that the execution time of the improved LZW algorithm is on the same scale as that of the Huffman algorithm. Additionally, it implies that the improved LZW algorithm has some advantages over the Huffman algorithm in terms of execution time.
(d) The improved LZW algorithm has a poor compression effect, but its execution time is shorter than that of LZSS1.

5. Conclusions

The goal of data compression technology is to reduce the volume or size of data by reducing redundant information and taking advantage of statistical properties in the data while maintaining the important content and integrity of the data. Data compression can not only help save storage space and reduce storage costs, but also improves the speed and efficiency of data transmission and reduces network traffic and transmission latency. By sacrificing data accuracy and introducing data compression technology, we can reduce the amount of communication data. The lossless data compression algorithm is chosen as the algorithm of the bridge inspection system and is further improved. In addition, through the comparison of the algorithm compression rates and execution times of the Huffman, LZSS, and improved LZW algorithms, we found that the improved LZW algorithm has strong adaptability, can meet any input data, and can fully use the repeatability of data, so it is very suitable for WSNs.

Acknowledgments

This work was supported by the National Natural Science Foundation of China (No. 61473329), the Fujian Provincial Natural Science Foundation of China (No. 2021J011235), and the phased research result of the provincial major project of the research project on education and teaching reform of undergraduate colleges and universities in Fujian Province [Project name: Research on the innovation education project of Internet of Things Engineering (Project No. fbjg202101018)].

References

1 S. Lykov, Y. Asakura, and S. Hanaoka: Tran. Res. Pro. **21** (2017) 56. https://doi.org/10.1016/j.trpro.2017.03.077
2 H. Tonny, Z. M. Hafiz, and P. A. N. Che: Multimedia Tools Appl. **82** (2022) 1. https://doi.org/10.1007/S11042-022-14130-1
3 X. Liu, Wang, B. Xu, K. Zheng, and X. Yao: Wireless Commun. Mobile Comput. **2022** (2022) 5. https://doi.org/10.1155/2022/2532905

4 B. Wang, H. Zhang, Z. Liu, and H. Xiao: WTC **2021** (2021) 1. https://doi.org/10.26914/c.cnkihy.2021.010444
5 D. Ning, W. Hao, G. Xu, J. Wan, and I. Muhammad: Ent. Inf. Sys. **2019** (2019) 1. https://doi.org/10.1080/17517575.2019.1633689
6 K. Xia, J. Liu, W. Li, P. Jiao, Z. He, Y. Wei, F. Qu, Z. Xu, L. Wang, X. Ren, B. Wu, and Y. Hong: Nano Energy **105** (2023) 1. https://doi.org/10.1016/J.NANOEN.2022.107974
7 P. Delgosha, and V. Anantharam: IEEE Trans. Information Theory **99** (2020) 1. https://doi.org/10.1109/tit.2020.2991384
8 Y. Song, Z. Zhu, W. Zhang, L. Guo, X. Yang, and H. Yu: Nonlinear Dyn. **95** (2019) 2235. https://doi.org/10.1007/s11071-018-4689-9
9 L. Wang, S. Li , S. Wang, D. Kong, and B. Yin: IEEE Trans. Mulimedia **23** (2021) 2857.https://doi.org/10.1109/TMM.2020.3017916
10 S. Justin and B. J. Pablo: IEEE Signal Process Lett. **24** (2017) 279. https://doi.org/10.1109/lsp.2017.2657381
11 W. Dong, G. Shi, X. Li,M. Yi, and F. Huang: IEEE Signal Proc. Soc. **23** (2014) 3618. https://doi.org/10.1109/TIP.2014.2329449
12 W. Qin, H. Wang, F. Zhang, J. Wang, X. Luo, and T. Huang: IEEE Trans. Image Process. **31** (2022) 2433. https://doi.org/10.1109/TIP.2022.3155949
13 D. Wang, G. Zhao, H. Chen, Z. Liu, L. Deng, and G. Li: Neural Networks **144** (2021) 320. https://doi.org/10.1016/J.NEUNET.2021.08.028
14 C. Wang, D. Xiao, H. Peng, and R. Zhang: J. Visual Commun. Image Represent. **51** (2018) 122. https://doi.org/10.1016/j.jvcir.2018.01.007J
15 Lee, S. Yoon, and E. Hwang: Sensors **21** (2021) 1521. https://doi.org/10.3390/S21041521
16 W. X. Zhao, X. Zhang, D. Lemire, D. Shan, J. Nie, H. Yan, and J. Wen: ACM Trans. Inf. Syst. **33** (2015) 1. https://doi.org/10.1145/2735629
17 R. Tom, and K. Shrinivas: IEEE Commun. Mag. **56** (2018) 28. https://doi.org/10.1109/mcom.2018.1700839
18 K. N. Shamimi, K. Amirrudin, P. L. Yee, and R. Hameedur: IEEE Access **6** (2018) 8011. https://doi.org/10.1109/access.2018.2796585

About the Authors

Zhengsong Ni received his bachelor's degree from Fuzhou University in 1995, his master's degree from Beijing Information Science and Technology University in 2007, and his doctorate degree from Beijing University of Posts and Telecommunications in 2010. From 2010 to 2012, he was a lecturer at Tianjin Polytechnic University, from 2012 to 2014, he was an assistant professor at Tsinghua University, and since 2014, he has been an associate professor at Fujian Normal University of Technology. His research interests include MEMS, big data, and sensors. (460532802@qq.com)

Shuri Cai received his bachelor's degree from Fujian Normal University in 1997 and his master's and doctoral degrees from Beijing University of Posts and Telecommunications in China in 2004 and 2008, respectively. Since 2007, he has worked as an associate researcher at the Institute of Highway Science under the Ministry of Transport. His research interests include MEMS, big data, and sensors. (caishuri@126.com)

Cairong Ni received her bachelor's degree from Sunshine College in 2022. She has been working as a teaching assistant at Fujian Normal University of Technology since 2022. Her research interests include MEMS, big data, and sensors. (3247146792@qq.com)