Performance Evaluation of Data Compression Algorithms for IoT-Based Smart Water Network Management Applications

Kazeem B. Adedeji*
Department of Electrical & Electronics Engineering, The Federal University of Technology Akure, Ondo State Nigeria

Abstract

IoT-based smart water supply network management applications generate huge volume of data from the installed sensing devices which are required to be processed (sometimes in-network), stored and transmitted to a remote centre for decision making. When the volume of data produced by diverse IoT smart sensing devices intensify, processing and storage of these data begin to be a serious issue. The large data size acquired from these applications increases the computational processing times, occupies the scarce bandwidth of data transmission and increases the storage space. Thus, data size reduction through the use of data compression algorithms is essential in IoT-based smart water network management applications. In this paper, the performance evaluation of four different data compression algorithms used for this purpose is presented. These algorithms, which include RLE, Huffman, LZW and Shanon-Fano encoding were realised using MATLAB software and tested on six water supply system data. The performance of each of these algorithms was evaluated based on their compression ratio, compression factor, percentage space savings, as well as the compression gain. The results obtained indicated that the LZW algorithm shows better performance based on the compression ratio, compression factor, space savings and the compression gain. However, its execution time is relatively slow compared to the RLE and the two other algorithms investigated. Most importantly, the LZW algorithm has a significant reduction in the data sizes of the tested files than all other algorithms.

Keywords: *background leakage, critical pipe, pressure reducing valve, water distribution network, water loss.*

1. Introduction

We are at the dawn of the 21st century, whereby several key events in technological improvements and research studies have materialised. One of the emerging research areas in recent years is the internet of things (IoT). As its name implies, several smart devices can be coordinated and connected via the internet. IoT is a system of web-enabled smart devices and sensors equipped with embedded processors and communication technology to acquire, transmit and process the acquired data from an environment [1]. Due to its seamless monitoring capabilities, it has been utilised in water supply network applications such as leakage monitoring, pipeline health monitoring, water quality monitoring, among others. In IoT-based smart water network management (SWNM) applications illustrated in Figure 1, real-time continuous measurement, which relates to the state of the network is

* Corresponding author. Tel.: +234-805-918-7995
E-mail address: kezman0474@yahoo.com; kezmantech4@gmail.com

Manuscript History:
Received 15 May, 2020, Revised 09 September, 2020, Accepted 11 September, 2020, Published 30 October, 2020

required. The data must be continuously acquired, processed and transmitted via wireless means to a remote centre where further analysis takes place for optimal decision making. Modern smart sensing devices have in-built data storage and processing capabilities, thus, in-network processing of data can be achieved [2, 3]. In each of the IoT-based applications for SWNM, huge volume of data, in terms of size and number, is generated. Unfortunately, the data acquired by the sensing devices are analogue, which are transformed into digital format via analogue-to-digital conversion (processing) before transmitting over wireless media. When the data are in the digital format, they are indicated as bytes/bits and the bit volume could be enormous, making processing tedious. The acquired water network data such as pressure (from pressure sensor), flow (from flow meters), sound (from acoustic sensors) etc. are converted into a digital stream comprising numerous bits of data. Large data size requires several bits to represent them, thus, during data pre-processing, the data are processed bit-by-bit. Data compression enables bit reduction to take place. Consequently, the number of bits required to be processed is reduced which enhances data computation. Therefore, due to the large volume of continuous data generated during smart water network applications, savaging data storage as well as reducing computational processing times through data size reduction, could further make IoT-based technology a better choice in water network management applications.

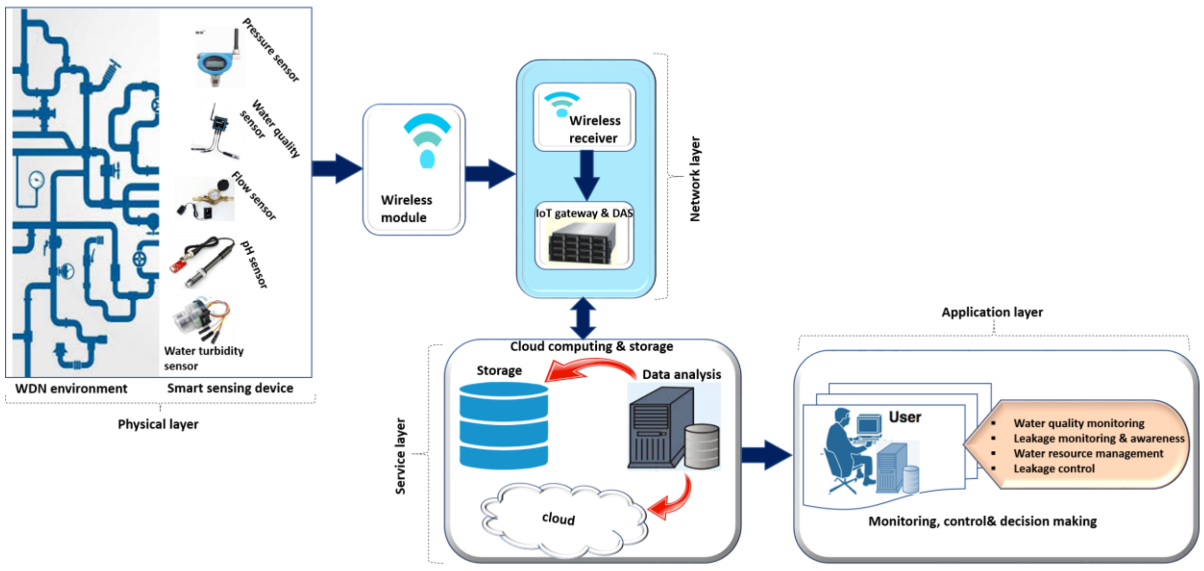

Figure 1: SWNM application framework.

Modern smart sensors are developed to perform data pre-processing tasks to reduce the load on gateways and cloud resources. These sensors permit the conversion of the physical variable measured into a digital data stream for transmission to a gateway. The basic building block of a typical smart sensor is shown in Figure 2. Smart sensing is developed to perform sensing, computation/data processing and communication tasks. The former involves the acquisition of the water network data while the second task is where the smart sensing device performs some analysis on the acquired data while the last task is to transmit the pre-processed data to a remote node. As shown in Figure 2, a smart sensor is equipped with a sensing unit, signal condition unit, an analogue to digital conversion (ADC) unit, storage and communication unit, amongst others. Once the water network data is acquired, these data are converted to digital stream within the ADC unit. In this unit, data compression takes place to reduce the data size for the storage unit and before transmission via the internet (a function of the communication unit or transceiver). As previously mentioned, data size reduction will have a significant effect on the computational processes of most data analysis performed by the smart sensors. These sensors are battery-powered, thus, enhancing the computational process by reducing the computational burdens which extends battery run-down time. Strictly speaking, computation is one of the major sources of energy consumption in modern smart sensors even though more energy is consumed during the communication task. The energy consumed during wireless transmission of a single bit of data may be about a thousand times more than that used for the computation of a single 32-bit of data [4]. For transmitting a single bit of data, energy is consumed, thus the more the number of bits to be transmitted, the more the energy consumed. Therefore, bit reduction will significantly reduce the energy consumption rate. This bit reduction is known as data compression. Many onboard processing hardware is equipped with data compressing capabilities (with a good compression algorithm) to reduce the computational burdens.

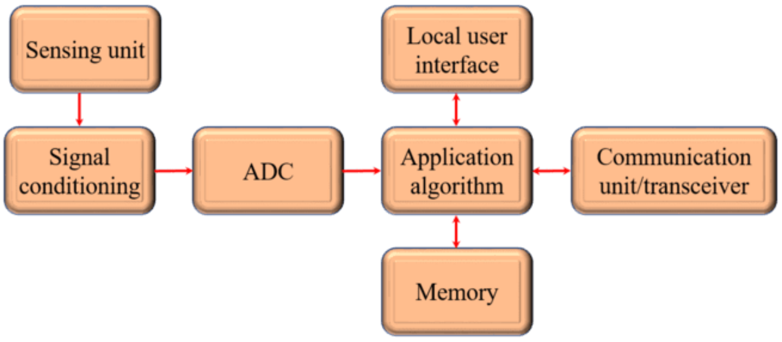

Figure 2: Basic building block of typical smart sensors [5].

Within the SWNM framework as illustrated in Figure 1, and as the need for application efficiency is increased, it is required that a plethora of smart sensing devices are utilised to acquire water network data. Thus, several MB/GB of the acquired data must be processed and transmitted to a remote centre via a communication network. It is well known that a large size data consumes bandwidth during transmission over the network. Unfortunately, bandwidth is limited and must be used effectively, one of the reasons why bandwidth optimisation research is frequently being conducted. Therefore, reducing the data size before transmission over the network is a better way of using optimizing bandwidth. This, together with the computing resource storage requirement due to increasing SWNM applications, have made the study of compression techniques within the SWNM framework crucial. In view of this, the performance of some compression techniques is analysed for their application within the SWNM framework. There are limited research papers within this framework, and therefore this study could be of importance to water utilities and research communities. It is expected that utilising one or a combination of data compression algorithms will improve transmission bandwidth as well as permit optimal use of the storage capacities. In terms of the storage capacities, most IoT-based systems have cloud storage features, thus stressing the fact that optimum use of storage is crucial. This, among other reasons, makes data compression an important research area for IoT vision. In the past and in recent times, data compression algorithms have been used in many applications [6-16]. One of the notable applications is telemedicine [8-12]. Nowadays, IoT systems are currently being applied to critical infrastructure monitoring [15, 16]. A typical example of such application is the water supply network monitoring. Motivated by this, the performance evaluation of some data compression algorithms for IoT-based water network management applications is presented. The article is arranged as follows: Section 2 briefly discusses the types of data compression, Section 3 presents the methodology, overview of the algorithms and the metrics used to assess the performance of each algorithm, Section 4 discusses the results obtained from the assessment, and finally, Section 5 presents the conclusion and future work.

2. Background: Data Compression

Data compression is a method utilised to reduce the number of bits required to express data. This method can save space, accelerate file transmission, and reduce transmission bandwidth and hardware storage costs [3]. Data compression can either be lossless or lossy. The former allows a file to be restored exactly to its initial state without losing a single bit of data during decompression. Considering the second type, it permanently removes irrelevant bits of data from the initial data. Thus, such type of data compression is used in applications such as images, where the elimination of some data bits is considered to have no significant impact on the data content. The lossless compressions are used in situations where the initial and decompressed data must be the same, or where a shift from the initial data would have a significant effect on the result. For instance, the data generated from the water supply system monitoring application must be perfectly reconstructed after compression without loss of information. Any loss of information or loss of numbers in the sensor readings would change the information which would significantly affect the analysis and decision making. For this reason, the focus of this study is on the use of lossless data compression. There exist varieties of lossless data compression as indicated in Figure 3 ranging from run-length encoding (RLE) to entropy-based coding. In this study, the performance of the RLE, LZW, Huffman and Shanon-Fano algorithms (selected from each sub-class as shown in Figure 1) was evaluated. These algorithms, will, of course, yield varying results, but all rely on the same fundamental principle of removing redundancy from the original data.

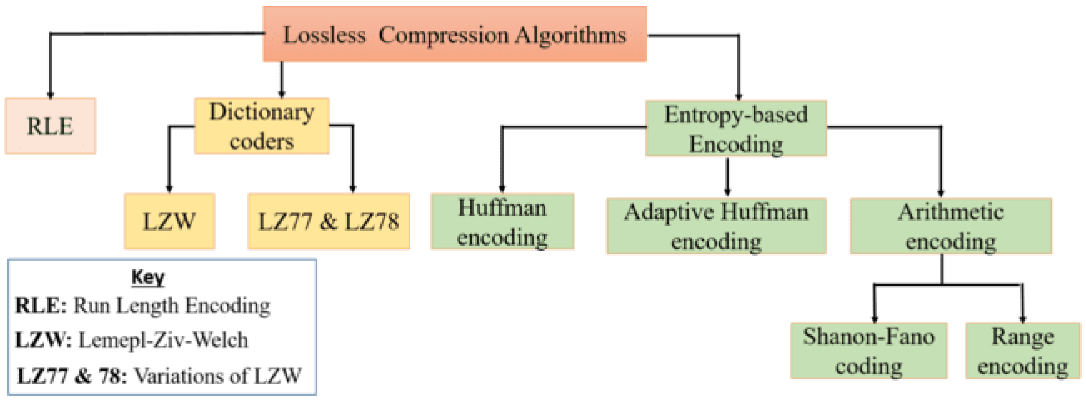

Figure 3: Types of lossless data compression algorithm.

3. Research Method

This work involved examining the performance of some data compression algorithms namely RLE, LZW, Shanon-Fano and Huffman coding, for IoT-based applications in water supply networks. These algorithms were executed in MATLAB software environment and tested on some water supply network data of varying sizes. Thereafter, the performance of each algorithm was evaluated based on their compression ratio, percentage space savings, compression factor and compression gain. An overview of the operation involved in some of these algorithms is first presented.

3.1 Overview of the data compression algorithms

3.1.1 LZW algorithm

LZW is a popular data compression algorithm named after the inventors Abraham Lempel, Jakob Ziv, and Terry Welch [17]. It operates by reading-in a series of symbols and arranging them into strings, before transforming the strings into codes. During the compression process, variables; CHAR and STR were utilised. CHAR holds a single character (that is, a single byte value between 0 and 255) while STR captures a group of one or more characters. Each character in the STR has a single byte. As shown in Figure 4(a), the algorithm begins by taking in the first byte of the input file and storing it in the STR. Thereafter, looping each additional byte of the input file commences. Subsequent reading of bytes from the input file is stored in the CHAR, thus creating a data table. This table is scanned to ascertain whether a code has already been designated to the concatenation of STR+CHAR. It only outputs a code for the STR when a match in the table is not spotted. Otherwise, the concatenation of STR+CHAR is stored in the STR, without further action.

3.1.2 Huffman Encoding

This is a well-known algorithm used for lossless compression of data. As shown in Figure 4(b), upon estimating the likelihood of each symbol, it creates a complete binary tree for different symbols and positions it in descending order. Huffman encoding uses variable-length encoding where all the characters are given a variable-length code based on how often they occur in the text [15]. The most frequently occurring character receives the smallest code while the least frequent ones get the largest code. The details of the operational process may be found in [13].

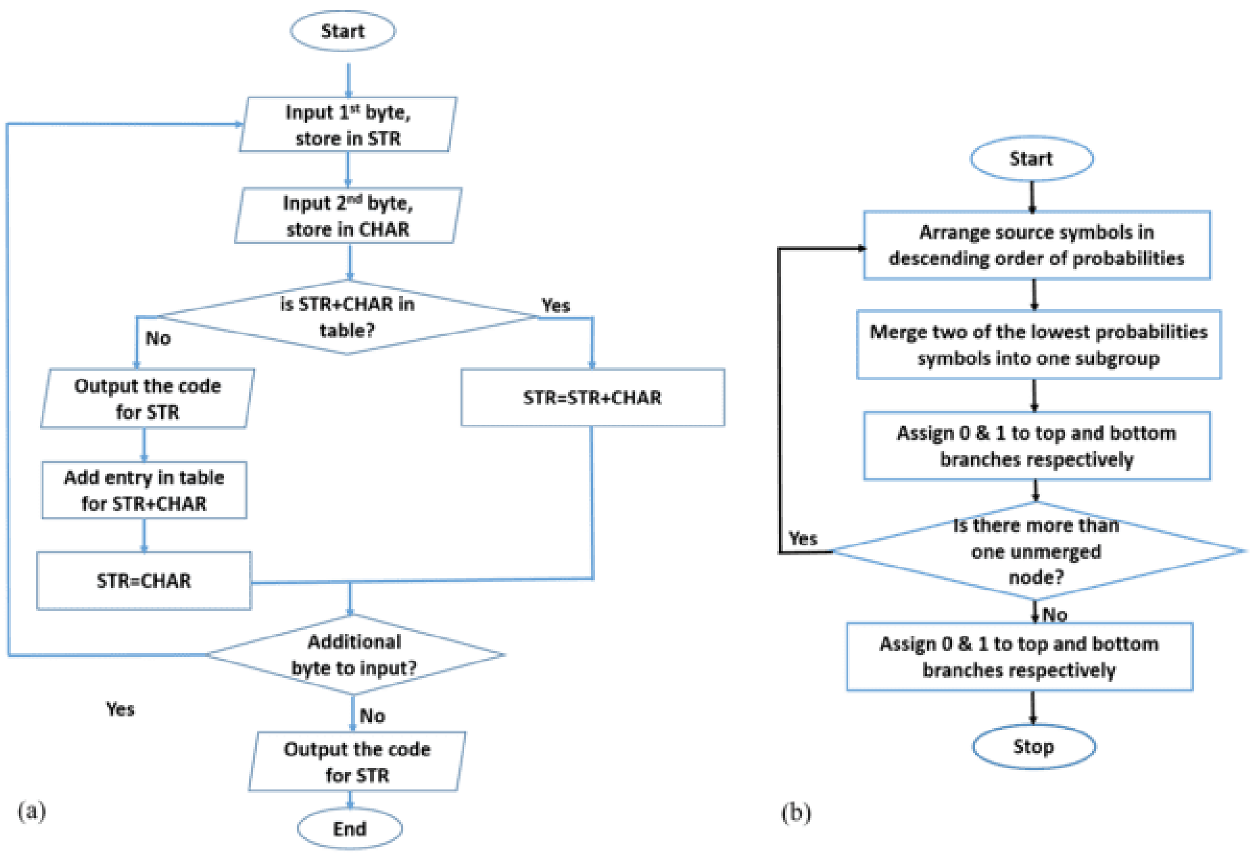
Start
Input 1st byte, store in STR
Input 2nd byte, store in CHAR
is STR+CHAR in table?
No
Yes
Output the code for STR
STR=STR+CHAR
Add entry in table for STR+CHAR
STR=CHAR
Additional byte to input?
Yes
No
Output the code for STR
End
(a)
Start
Arrange source symbols in descending order of probabilities
Merge two of the lowest probabilities symbols into one subgroup
Assign 0 & 1 to top and bottom branches respectively
Is there more than one unmerged node?
Yes
No
Assign 0 & 1 to top and bottom branches respectively
Stop
(b)

Figure 4: Flow chart for the lossless data compression algorithms (a) LZW (b) Huffman coding [13].

3.1.3 RLE

This is a simple and common algorithm based on the idea of substituting a long sequence of the same symbol by a shorter one. As shown in Figure 5(a), *runs* of data, which is the sequences at which similar data values occurring in successive data elements are stored as a single data value and *count* rather than as the original *run*. This method minimises the rate of replicating symbols in a string through the use of a special marker at the start of the symbol. The details of the operational process may be found in [18].

3.1.4 Shannon-Fano

This is an entropy-based encoding technique which utilises a variable-length code for encoding a source symbol. In a similar manner to the Huffman coding, it generates a frequency table and then divides this into the upper and lower part. During operation, it ensures that either of these parts has nearly the same sum of frequencies [18]. This process is repeated until only a single symbol is left. The flow chart of the Shannon-Fano lossless data compression algorithm is illustrated in Figure 5(b).

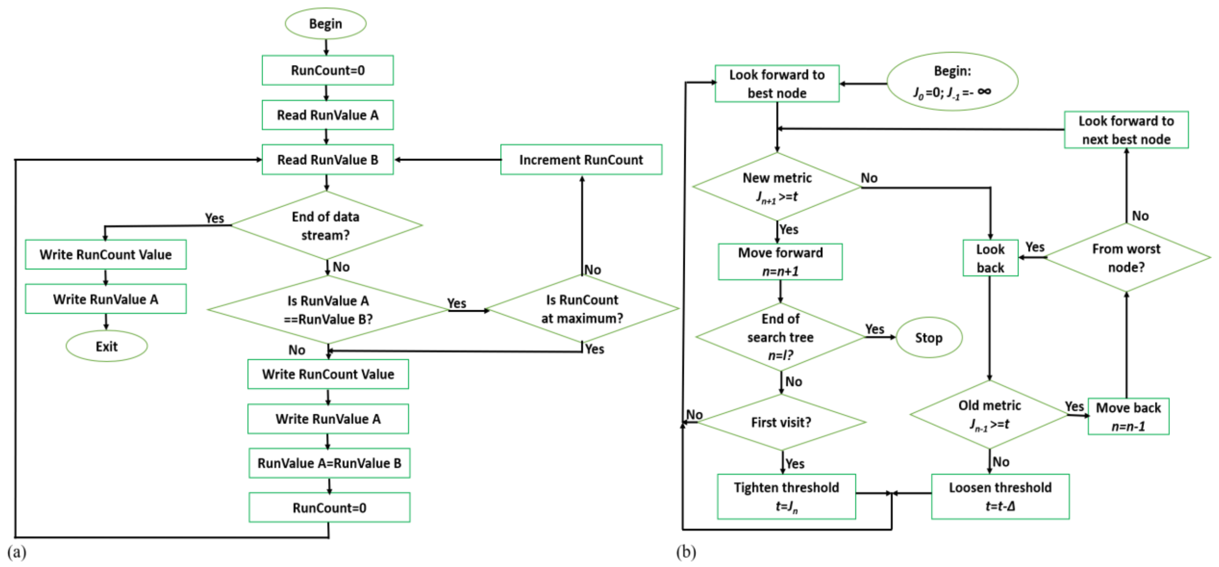

Figure 5: Flow chart for lossless data compression algorithms (a) RLE (b) Shanon-Fano algorithm [18].

3.2 The algorithm evaluation

Each of these algorithms with the flow chart displayed in Figure 4 and Figure 5 were used to compress and decompress six case study test data having properties shown in Table 1. The description of each data file was reported in Table 1. These data are derived from a water supply network reporting pressure, flow and leak discharge in the network within a specific period. For example, the test file 1 which consists of a minimum night flow (MNF) data reports the flow measurements within the water supply network obtained between the hours of 2:00 am and 4.00 am. As mentioned earlier, these data were used to test the data compression algorithms. The execution times for both compression and decompression processes were recorded in each case. The simulation was performed in MATLAB software environment on a system with an Intel(R) Core i7-2620M, 2.7 GHz processor with Windows 10.1 platform. Thereafter, the performance of the algorithms was evaluated by estimating the performance indices discussed in sub-section 3.3.

Table 1: Properties of the test file data used as case studies.

S/N	Test file	Size (*byte*)	Description
1	Test file 1	9524	Minimum night flow data
2	Test file 2	12342	24 hours water flow data within the piping network
3	Test file 3	15340	24 hours leak discharge data.
4	Test file 4	47102	24 hours pressure data within the piping network
5	Test file 5	51211	48 hours water flow data
6	Test file 6	95501	1 year water consumption data

3.3 Performance evaluation index

Several indices can be utilised to analyse the performance of these algorithms. In this study, the compression ratio, percentage space savings, compression factor, and compression gain were used. Among these, the compression ratio is the most frequently used [19]. The compression ratio describes the average number of bits needed to store the compressed data. The compression ratio *CR* is expressed as

$$CR = \frac{N_c}{N_{unc}} = \frac{S_c}{S_{unc}} \tag{1}$$

where N_c and N_{unc} depict the number of bits in the compressed and uncompressed (original) data respectively, S_c is the size of the compressed data while S_{unc} is the size of the original data. The compression factor *Cf* is another measure used to assess the performance of these algorithms. The *Cf* is seen as the inverse of the *CR*. It is expressed as

$$Cf = \frac{1}{CR} = \frac{N_{unc}}{N_c} = \frac{S_{unc}}{S_c} \tag{2}$$

Since compression algorithms permit optimum use of storage, it is important to assess them based on the space-saving aspect. Thus, percentage space savings (*%SS*) is also considered. This index, expressed in equation (3), defines the reduction in file size relative to its uncompressed size [19].

$$\%SS = \left(1 - \frac{N_c}{N_{unc}}\right) \times 100\% = \left(\frac{S_{unc} - S_c}{S_{unc}}\right) \times 100\% \tag{3}$$

This study also considers the compression gain (*CG*) of each algorithm. This is expressed in equation (4).

$$CG = 100 \log_e \left(\frac{S_{unc}}{S_c}\right) = 100 \log_e \left(Cf\right) \tag{4}$$

Even though the *CR* and *Cf* are sufficient to assess the performance of lossless data compression algorithms as reported in [19], other indices described above were also analysed. It is noteworthy to stress that for lossy compression algorithms, where the decompressed data varies slightly from its initial form, more performance measures are required to appraise the level of distortion, fidelity and quality of the data such as an image.

4. Results and Analysis

Each of the four lossless compression algorithms is executed in MATLAB software programme and tested on six different data of varying sizes. Figure 6 shows the compression ratio and the compression factor for each algorithm. In Figure 6(a), it was observed that both LZW and Huffman algorithms had the lowest *CR*. This means that fewer bits were required to store the compressed file using these algorithms. It was observed in Figure 6(a) that the *CR* for the LZW was better than the other three algorithms. Furthermore, analysing Figure 6(a) considering the test file 1, the LZW had a *CR* of 0.48. This implied that it could reduce the data size by 52% while RLE could achieve a reduction capacity of 28%. The reduction capacity for Shanon-Fano and Huffman coding was observed to be 38% and 44% respectively considering the test file 1. The *Cf* results (Figure 6(b)) also showed that the LZW algorithm, having the highest *Cf* performed better than the other algorithms. Figure 7 illustrates the percentage space savings and the compression gain of the compression algorithms. In Figure 7(a), it was observed that both LZW and Huffman algorithms had the highest space savings while RLE had the lowest. As noticed for all the test files, the LZW algorithm achieved better space-saving. The results of the compression gain also indicated that the LZW algorithm had better performance than the other three algorithms tested on the test files.

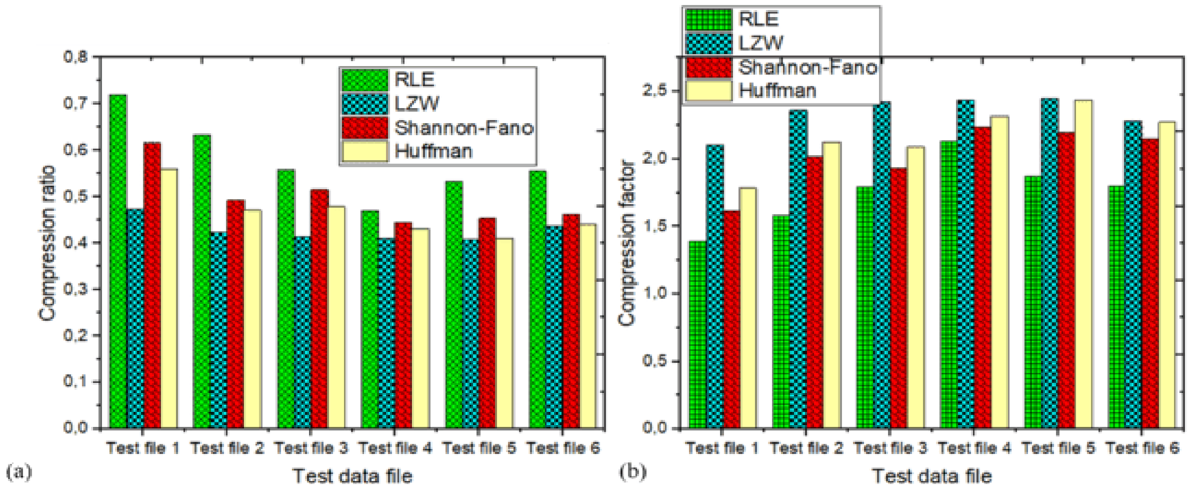

Figure 6: Plot of the algorithm performance based on (a) compression ratio and (b) compression factor.

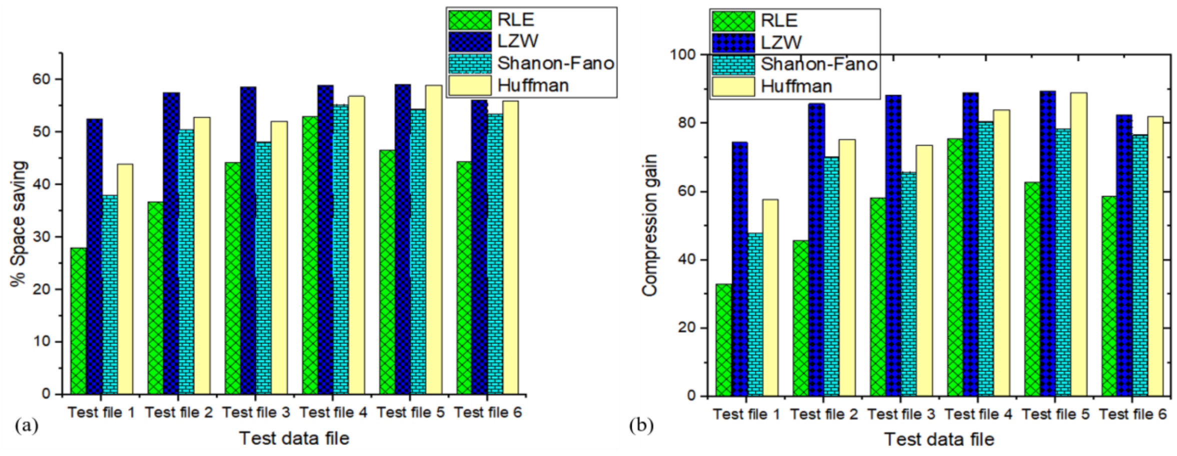

Figure 7: Plot of the algorithm performance considering (a) % space savings (b) compression gain.

The results in Table 2 shows the comparison of the compressed file size as well as the execution times for the four lossless data compression algorithms. Considering the file size reduction, LZW had a better file size reduction followed by Huffman coding. Table 2 and test file 1 with size 9524 *bytes* show that the LZW algorithm produced a compressed file with size less than half the original file size. The same result was noticed when test file 2 to 6 were considered. In respect of the three other algorithms, that is, RLE, Shanon-Fano and Huffman, the compressed file sizes were more than half the original file sizes for all the test files. Amongst these algorithms, the RLE produced the least performance. In terms of the execution time in milliseconds (*ms*), it was observed that RLE was relatively faster during both compression and decompression processes than the other three algorithms for the same data size. For all the four algorithms, it was observed that it took more time to decompress the data files. In most cases, the decompression time (DT) was observed to be more than twice the compression time for all the algorithms. The execution time, will, however, depend on the CPU structure and the RAM size of the computing system used to perform the test.

Table 2: Comparison of the compressed file sizes and execution times for each algorithm.

Original File size (*bytes*)	**Compressed file size (*bytes*)**				**RLE**		**LZW**		**Shanon-Fano**		**Huffman**	
	RLE	**LZW**	**Shanon-Fano**	**Huffman**	**CT**	**DT**	**CT**	**DT**	**CT**	**DT**	**CT**	**DT**
9524	6852	4521	5891	5341	41	110	42	123	34	150	35	144
12342	7812	5231	6101	5810	40	125	73	131	42	180	60	172
15340	8563	6341	7932	7342	50	140	80	142	53	192	69	163
47102	22121	19321	21022	20322	59	180	130	170	62	480	128	310
51211	27342	20894	23321	21011	81	301	172	470	120	510	92	410
95501	53041	41832	44321	42013	143	312	169	310	230	710	159	601

CT: Compression time (*ms*); **DT:** Decompression time (*ms*)

4. Conclusions

In this paper, the performance of four different lossless data compression algorithms for IoT-based smart water network management application is presented. The results obtained showed that the LZW algorithm had a better performance in terms of compression ratio, compression factor, space savings and the compression gain. However, its execution time was relatively slow compared to RLE and the two other algorithms investigated. In general, the LZW algorithm gave a significant reduction in the data sizes of the tested files compared to all other algorithms. As the future of the IoT-based smart water management applications grow, the application requirements will also be heightened necessitating the further development of incorporated smart sensors. Thus, more volume of data will be produced by the various IoT smart sensing devices which makes storage, transmission bandwidth and data processing challenging. Future research studies should consider the development of a hybrid data compression matrix that will balance the trade-off between space savings and the speed of compression for applications that require real-time continuous data analysis. This will alleviate data bottleneck issues.

References

[1] Adedeji, K.B., Nwulu, N and Aigbavboa, C. (2019). IoT-based smart water network management: Challenges and future trend. *In: Proceedings of the IEEE Africon Conference*, September 25–27, Accra, Ghana.

[2] Hassanalieragh, M., Page, A., Soyata, T., Sharma, G., Aktas, M., Mateos, G., Kantarci B. and Andreescu S. (2015). Health monitoring and management using IoT sensing with cloud-based processing: Opportunities and challenges, *In: Proceedings of the 2015 IEEE International Conference on Services Computing*: 285–292.

[3] Al-Turjman, F. and Alturjman S. (2018). Confidential smart-sensing framework in the IoT era, *The Journal of Spercomputing*, Vol. 74, No. 10, 5187–5198.

[4] Barr, K.C., and Asanović, K. (2006). Energy-aware lossless data compression, *ACM Transactions on Computing Systems*, Vol. 24, 250–291.

[5] Tech Briefs (2018). Smart sensor technology for the IoT, Tech Briefs Magazine, Engineering Solution for Design & Manufacturing, Vol. 42, No. 11. www.techbriefs.com.

[6] Deepu, C.J., Heng, C.H. and Lian, Y. (2016). A hybrid data compression scheme for power reduction in wireless sensor for IoT, *IEEE Transactions on Biomedical Circuits and Systems*, Vol. 11, No. 2, 245–254.

[7] Hwang, W.J., Chine, C.F. and Li, K.J. (2003). Scalable medical data compression and transmission using wavelet transform for telemedicine applications, *IEEE Transactions on Information Technology in Biomedicine*, Vol. 7, No. 1, 54–63.

[8] Antonopoulos, C.P. and Voros, N.S. (2016). Resource efficient data compression algorithms for demanding, WSN based biomedical applications, *Journal of Biomedical Informatics*, Vol. 59, 1–4.

[9] Lucas, L.F., Rodrigues, N.M., da Silva, C.L. and Faria, S.M. (2017). Lossless compression of medical images using 3-D predictors, *IEEE Transactions on Medical Imaging*, Vol. 36, No. 11, 2250–2260.

[10] Reddy, B.V., Reddy, P.B, Kumar, P.S. and Reddy, A.S. (2016). Lossless compression of medical images for better diagnosis, In: *2016 IEEE 6th International Conference on Advanced Computing*, Feb., 27, 404–408.

[11] Kumar, V., Saxena, S.C. and Giri, V.K. (2006). Direct data compression of ECG signal for telemedicine, *International Journal of System Science*, Vol. 37, No. 1, 45–63.

[12] Ayinde, B.O. (2017). A fast and efficient near-lossless image compression using zipper transformation, arXiv preprint arXiv:1710.02907: 1–13.

[13] Mahmud, S. (2012). An improved data compression method for general data, *International Journal of Scientific and Engineering Research*, Vol. 3, No. 3, 1–4.

[14] Boban, A. and Vladan, V. (2018). Efficient image compression and decompression, *Electronics and Energetics*, Vol. 31, No. 3, 461–485.

[15] Mohamed, M.I, Wu, W.Y. and Moniri, M. (2013). Adaptive data compression for energy harvesting wireless sensor nodes, In *proceedings of the 10th IEEE international conference on networking, sensing and control*, April 10, 633v638.

[16] Richard, J., Heiko, M. and Veit, H. (2018). Comparison of lossless compression scheme for high rate electrical grid time series for smart grid monitoring and analysis, *Computers and Electrical Engineering*, Vol. 71, 465–476.

[17] Ziv, J and Lempel, A. (1977). A universal algorithm for sequential data compression, *IEEE Transactions on Information Theory*, Vol. 23, No. 3, 337v343.

[18] Kavitha, P. (2016). A survey on lossless and lossy data compression methods, *International Journal of Computer Science & Engineering Technology*, Vol. 7, No. 3, 110–114.

[19] Uthayakumar, J., Vengattaraman, T. and Dhavachelvan, P. (2019). Survey on data compression techniques: From the perspective of data quality, coding schemes, data type and applications, *Journal of King Saud University –Computer and Information Sciences*. In press.