1. Astrit HULAJ[1], 2. Bahri PREBREZA[2*], 3. Xhevahir BAJRAMI[2]

University of Business and Technology (1), University of Prishtina (2)
ORCID: 1. 0000-0002-4059-2034; 2. 0000-0003-1950-026X; 3. 0000-0002-6901-0122

doi:10.15199/48.2023.12.20

Compressing big data generated by IoT devices deployed along the green borderline

***Abstract**. Deployment of the thousands of sensors along the green borderline the production of a large volume of data. The large volume of data can lead to unusual storage and transmission bandwidth requirements. The research presents an analytical and visual analysis of the performance of the data compression algorithms. The algorithms are modified and adapted for our research purpose, and the results are visualised graphically. This research includes a detailed analysis of data compression algorithms and will optimise the overall cost of storing data, and the link capacity.*

***Streszczenie.** Wdrożenie tysięcy czujników wzdłuż zielonej granicy produkcji dużej ilości danych. Duża ilość danych może prowadzić do nietypowych wymagań dotyczących przepustowości pamięci masowej i transmisji. W pracy przedstawiono analityczną i wizualną analizę działania algorytmów kompresji danych. Algorytmy są modyfikowane i dostosowywane do naszych celów badawczych, a wyniki są wizualizowane graficznie. Badania te obejmują szczegółową analizę algorytmów kompresji danych i zoptymalizują całkowity koszt przechowywania danych oraz przepustowość łącza. (**Kompresja dużych zbiorów danych generowanych przez urządzenia IoT rozmieszczone wzdłuż zielonej granicy**)*

Keywords: Data compression, Internet of Things (IoT), Green borderline, Sensors.
Słowa kluczowe: Kompresja danych, Internet rzeczy (IoT), Zielona granica, Czujniki.

Introduction

Today Internet of Things (IoT) is a technological revolution. IoT refers to the billions of intelligent physical devices around the world that connect to the Internet and which can use to collect and share data. Devices in an IoT network represent an interface between the physical world and the world of electronic devices. These devices are sophisticated devices that can be used to detect or collect information in a physical environment, such as motion detection, temperature changes, image capture, pressure measurement, vibration detection, specific observations, etc.

The application of IoT smart devices is expanding rapidly [1]. Today, we find smart IoT devices in almost every sphere of life, such as smart healthcare applications [2, 3] smart military surveillance, smart transportation monitoring, smart homes, smart cities, etc. In this research, we will focus on the application of IoT for surveillance purposes of the state green border. Sensor devices have aroused considerable interest in many applications and research. In particular, our focus is on multimedia sensors. Multimedia sensors are devices that use a camera through which sensors provide a view of their monitoring area. Therefore, the application of multimedia sensors to surveillance of the green borderline will offer some benefits. However, the application of multimedia sensors for this purpose is characterized by several challenges that we have addressed in this research. The green borderline due to its specifics is very delicate and an actual issue in terms of national security. The protection of green state borders is about national security. Border protection against attacks and threats of various kinds is vital for a state. Therefore, the use of multimedia sensors will provide surveillance of the green state borderline based on image capture techniques. However, these detection techniques are characterized by several challenges. Some of the challenges are: security authorities need to place multimedia sensors in areas not easily accessible, limited capacity of the link to carry them, insufficient storage capacity for images captured by IoT devices, security of collected data from IoT devices, the lifetime of multimedia sensors directly depends on their battery, etc. Therefore, data compression will directly affect the reduction of data size. Reducing the data size will also impact the reduction of the requirements for link and storage capacity.

Problem description

States constantly have problems of various natures regarding illegal crossings along state borders. Border problems differ from one state to another, depending on the configuration of the state borderline that its state has. The borderline of a state means the borderline that separates the territory of that state from other neighbouring states [4]. This border is also a state border. The state border of a country can be divided into land borders, water borders, and air borders. The land border is also known as the "green border" and means any borderline between that country and neighbouring countries, excluding official crossing points. The green border in many countries is the most problematic borderline to the application of surveillance and security systems [4]. Not infrequently, the green borderline includes delicate areas difficult to reach by the security authorities and areas that can easily use for illegal border crossings. Illegal border crossings can use for various purposes, such as illegal immigration, terrorism, arms and drug smuggling, etc.

For securing the green borderline, the security authorities apply different techniques from conventional ones to the application of intelligent technology or IoT [5]. The application of these technologies along the state green line will characterize by some problems of various natures. Among these problems that can point out are the difficulties of deploying technology, coverage of the area with the Internet network, insufficient capacity of the transmission link, physical security of devices, electricity supply, etc. Many of these problems resolve with the application of 5G mobile technology. 5G technology offers unlimited data transmission capabilities within mobile operating end systems. The 5G technology has several features which other generations have not managed to contain as an extended range of movement, low connection establishment delay, greater transmission bandwidth, simultaneous connection of many devices, and highly reliable connection between different devices. In addition, 5G technology offers ultra-low latency of 1 ms, 90% more power efficiency, 99.9% reliability of connection establishment, and a maximum data transmission speed of 10Gbps.

One of the many IoT devices that have aroused interest for application for border security purposes is the multimedia sensor. Multimedia sensors are the future

technology that will impact solutions to many problems at a low cost [6]. The application of multimedia sensor technology to surveillance of the border green line will have multidimensional benefits. The low cost allows Figure 1 presents a possible IoT devices deployment architecture along a state border green line. The network architecture of IoT devices divides into three parts. The first part includes the green border line along which deployment of IoT devices. The second part includes the end buildings of the police stations, which are closest to the green border. The third part of the network architecture is the central monitoring room, which houses will be the data centre [6]. The area along the green borderline will cover by the Internet network through the microwave network, namely with 5G technology. The communication between the police stations, the monitoring centre, and the data centre can do through the optical or microwave network.

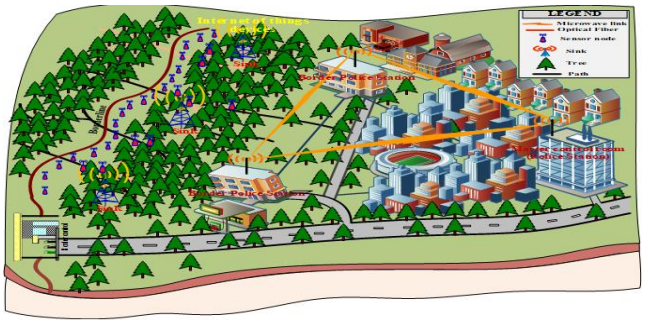

Fig. 1. The architecture of deployment of IoT devices along the green borderline [6]

As you can see, along the green borderline, there can be thousands of IoT devices, and each device transmits data from the area it covers. The data, which each device transmits to the data centre, must be stored and analyzed. Uncompressed storage of such a large volume of data requires very large resources and may be unaffordable for security authorities.

Research Methodology

The data collected along the green borderline by IoT devices, through 5G technology, are sent to the main centre for other analysis. The data collected by IoT devices need compression to save data storage capacity. For data compression that is collected, we have chosen to compare the performance of the three most popular algorithms: Huffman, LZ77, and Run-Length Encoding. First, we saved the images captured by the multimedia sensors in a file. For comparison and analysis of the performance of the algorithms, we chose this file which contains a maximum of 3600 image data. To make the performance measurement and algorithm comparison more attractive, we split this file into 11 other files through this partition: .jpg files with 1, 10, 100, 300, 500, 1000, 1500, 2000, 2500, 3000, and 36000 image data (the test remains the same for all algorithms across iterations). In [5, 6], you can read more about the types of sensors located along the green bounder line. We have written the algorithms in the Python programming language.

Implementation of algorithms

In this section, we will present the implementation of the algorithms and pseudo-codes. Algorithm 1 showed the pseudo-code of the Huffman algorithm. As seen in the pseudo-code presented in Algorithm 1, the Huffman encoding takes as input a set of n characters, present in pseudo-code with C character. Initially, all nodes are child nodes and contain the symbol, the frequency of the symbol, and optionally a connection to its child nodes. Bit '0' represents the left child while bit '1' represents the right child [7].

Algorithm 1. Pseudo code for the Huffman Coding Algorithm

```
procedure Huffman_Coding_Algorithm(Ch):
    n ← Ch.size_of_text
    Qu ← priority_queue()
    for i ← 1 to n
        n ← node_of_binary_tree(Ch[i])
        Qu.push_node(n)
    end for loop
    while Qu. size of_text() is not equal to 1
        Zu ← new node_of_binary_tree ()
        Zu.left ← x_l ← Qu.pop
        Zu.right ← y_r ← Qu.pop
        Zu.frequency ← x_l.frequency + y_r.frequency
        Qu.push(Zu)
    end while loop
    return Qu
end procedure
```

Priority queue Qu in pseudo-code is to hold nodes so that the node with the lowest frequency is extracted as the first node [7]. As long as there is more than one node in the queue, the two with the highest priority nodes are removed from the priority queue, and a new internal node Zu will create that has the two extracted nodes x_l and y_r as child nodes. The created internal node Zu now has a frequency equal to the sum of the frequencies of the two child nodes.

Algorithm 2. Pseudo code for the Lempel-Ziv LZ77 compression Algorithm

```
procedure Lempel_Ziv_LZ77_Algorithm (chars):
  while input is not empty do
    prefix← the longest prefix of input that begins in the
window
    if a prefix exists then
      dis← distance to start of prefix
      ln← length of the prefix
      char← the following prefix in the input
    else
      dis← 0
      ln← 0
      char← first char of input
  end if loop
    output (dis, ln, char)
    discard ln + 1 char from the front of the window
    s← pop ln + 1 char from the front of the input
    append s to the back of the window
  end while loop
end procedure
```

The new node created now will add to queue Qu in code. We have modified the code for the Huffman algorithm so that the output is not a printed string showing the encoding of the characters, but in the output, we get a list with two elements. The first element represents the number of bytes occupied by the decoding table, which stores both the character and the encoding of the characters. The second element represents the number of bytes of encoded text. If we were to create a format file for Huffman encoding, it would contain the table as a "header" and the content. In other words, the file would have a total size equal to the sum of these two.

Algorithm 2 presents the pseudo-code for the LZ77 algorithm. This algorithm starts by searching the window for the longest match at the start of the look-ahead buffer and outputs the pointer of that match [8]. As seen in the algorithm, the match pointer comes out as a triple of elements: dis - represents the offset, ln - represents the length of the match, and char - represents the next

character after the character match [9, 10]. If there is no match, the output of the algorithm is a null pointer (the offset and length of the character match will be: 0) [11], as well as the first character from the input. In the code for the LZ77 algorithm, we use the data compression class. For visualization, we wrote the method which opens the file with certain lines of data, reads it, and returns the number of bytes after compression.

Algorithm 3. Run-Length Encoding Algorithm

```
procedure Run_Length_Encoding_Algorithm(text)
    compressed = "data_input"
    i←0
    while i is less than the length(text)
        count ← 1
        while i + 1 is less than length(text) and text[i] is ~ to text[i+1]
            count ← count + 1
            i ← i + 1
        end while loop
        compressed_text = string(count) + text[i]
        i ←i + 1
    end while loop
    return compressed_text
end procedure
```

Algorithm 3 presents the pseudocode for the Run-Length Encoding Algorithm. Algorithm 3 shows that a string of characters is inserted as input, and an output will be obtained as a compressed text. For each input character, it is checked whether that input character is inside the text, and this string character is compared to the next string character if it is the same. With the fulfilment of these conditions, a value that we set as a counter of the number increases, and for each iteration, we add the number of repetitions and the repeated character to the string that will be output. For example, ABBCCCD on input to output gives the result 1A2B3C1D.

Results and discussion

This section presented the results of testing the algorithms and analysing their performance. After testing and analysing three compression algorithms: Huffman, LZ77, and Run-Length Encoding, we have presented and visualized the results graphically through Python libraries and the interactive Jupyter Notebook platform.

Space saving and execution time efficiency are the two most important factors for a compression algorithm. The performance of a compression algorithm depends to a large extent on the iterations in the data. To generalize the testing of algorithms, we have used the same data (image data) to test the three algorithms. We will compare the performance of the algorithms based on the following parameters: Compression ratio, compression factor, storage percentage, and compression time. The compression ratio is calculated through equation (1). Equation (1) gives the compression ratio between the compressed file and the original file (uncompressed file).

$$Compression\ ratio = \frac{Compressed\ data\ file}{Original\ data\ file} \tag{1}$$

The compression factor is calculated through equation (2). Equation (2) gives the ratio between the original and the compressed file and at the same time is the inverse of the compression ratio.

$$Compression\ fsctor = \frac{Original\ data\ file}{Compressed\ data\ file} \tag{2}$$

The storage save percentage is calculated through equation (3) and represents the percentage of file size equation (3) and represents the percentage of file size reduction after their compression.

$$Storage(\%) = (1 - \frac{Compressed\ data\ file}{Original\ data\ file}) \tag{3}$$

Execution time represents the time for which the algorithm has compressed the file. The test results of each algorithm, according to the measurement parameters, are given in the tables. The fields of each table are the number of lines of the image data, number of bytes in the original data, number of bytes in the compressed data, execution time in milliseconds, compression ratio, compression factor, and save percentage. We see in the results of the Huffman algorithm presented in Table 1 that we do not have any compression in the case of testing the algorithm with 1 and 10 lines of image data. But, the file size increased after the compression file.

Table 1. Huffman algorithm test results

Lines	OrigBytes	CompBytes	CompTimeMS	CompRatio	CompFactor	Saving%
1	53	349.375	0.9954	6.592	0.1517	-559.1981
10	422	943.625	0.9968	2.2361	0.4472	-123.6078
100	3608	3091.375	6.454	0.8568	1.1671	14.3189
300	12828	8430.375	18.5204	0.6572	1.5216	34.2815
500	20969	13110.375	10.5336	0.6252	1.5994	37.4773
1000	42314	25317.5	27.5657	0.5983	1.6713	40.1676
1500	61290	35961.25	24.436	0.5867	1.7043	41.3261
2000	80883	47344.75	65.3386	0.5853	1.7084	41.4651
2500	102664	59630.125	66.2417	0.5808	1.7217	41.9172
3000	122896	71246.125	115.8903	0.5797	1.7249	42.0273
3600	148574	85891.875	90.3051	0.5781	1.7298	42.1892

It is because the table which stores the characters and their encodings takes up extra space. So, for compressing so few rows, it is not worth creating this table. It does not mean that the algorithm does not work clearly for more lines of data to become acceptable data compression. For example, for 3600 lines of image data, a reduction of about 42.19% of the space occupied by the original data was made. The execution time for 3600 lines of image data is approximately 90.31 milliseconds or 0.0931 seconds. Similarly, in Table 2, we present the results after testing the LZ77 algorithm for different numbers of lines of image data. Similar to the Huffman algorithm we do not have any compression with 1 and 10 lines of the image data. But even in this case, there is an increase in the space required to store the data images. It happens (as we mentioned in Huffman's algorithm) because the table that stores the characters and their encodings takes up extra space. However, for more image data, for example, 3600 lines of the image data, we have a space reduction of about 37.17%. The execution time for 3600 image data is about 65454,495 milliseconds or 65.45 seconds.

Table 2. LZ77 algorithm test results

Lines	OrigBytes	CompBytes	CompTimeMS	CompRatio	CompFactor	Saving%
1	53	121	2.4912	2.283	0.438	-128.3019
10	422	567	64.595	1.3436	0.7443	-34.3602
100	3608	2759	768.8448	0.7647	1.3077	23.531
300	12828	8496	3266.0046	0.6623	1.5099	33.7699
500	20969	13697	5479.0359	0.6532	1.5309	34.6798
1000	42314	26894	13343.024	0.6356	1.5734	36.4418
1500	61290	38562	23038.435	0.6292	1.5894	37.0827
2000	80883	51190	30068.9254	0.6329	1.5801	36.7111
2500	102664	64523	41916.3013	0.6285	1.5911	37.1513
3000	122896	77396	51918.6287	0.6298	1.5879	37.0232
3600	148574	93345	65454.495	0.6283	1.5917	37.1727

In Table 3, we present the results after testing for the Run-Length Encoding algorithm. Unlike other algorithms, this algorithm for no test case, in other words, for any

number of image data, from 1 to 3600, does not offer compression in data size but only requires an addition to the space required to store these data. This case best tells us that the compression algorithm should be adapted to the data that is compression. It is because we can see from this case that the discrepancy between them can increase the space required by 100% (for example, in this case, compression of image data, the need for space for storing after data compression is doubled). The reason for these poor results is that in natural language data, we do not often come across words where we have the same characters one after the other. So, although a well-known and well useful algorithm for specific data compression, it does not provide good results in compressing data in the natural language. The execution time for 3600 image data for this algorithm is about 253.32 milliseconds or 0.25 seconds.

Table 3. Run-Length encoding algorithm test results

Lines	OrigBytes	CompBytes	CompTimeMS	CompRatio	CompFactor	Saving%
1	53	106	0	2	0.5	-100
10	422	816	6.5351	1.9336	0.5172	-93.3649
100	3608	6765	20.2198	1.875	0.5333	-87.5
300	12828	24177	46.8938	1.8847	0.5306	-88.4705
500	20969	39656	69.2239	1.8912	0.5288	-89.1173
1000	42314	80107	88.4743	1.8932	0.5282	-89.3156
1500	61290	115795	103.6453	1.8893	0.5293	-88.9297
2000	80883	153131	147.2414	1.8932	0.5282	-89.3241
2500	102664	194742	206.1033	1.8969	0.5272	-89.6887
3000	122896	232813	244.374	1.8944	0.5279	-89.439
3600	148574	281183	253.3214	1.8925	0.5284	-89.2545

The achieved results will also be presented in graphical form. For each algorithm, the results are presented through two types of graphs: a scatter graph and a bar graph. For each graph, the x-axis represents the number of data lines of images, while the y-axis represents the number of bytes. From the legend, we read the dots or columns and look at the compression level for each test. The data from Table 1 for the Huffman algorithm can be seen graphically in Figure 2 and Figure 3.

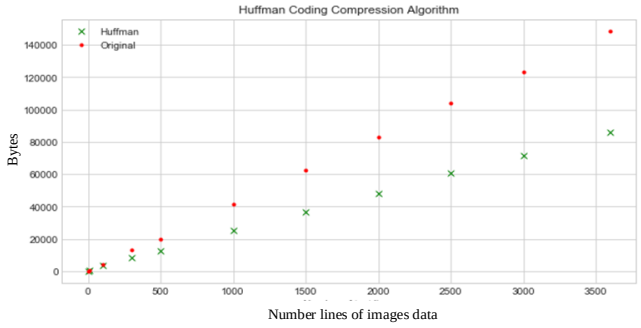

Fig.2. Graph (scatter) of results after testing the Huffman algorithm

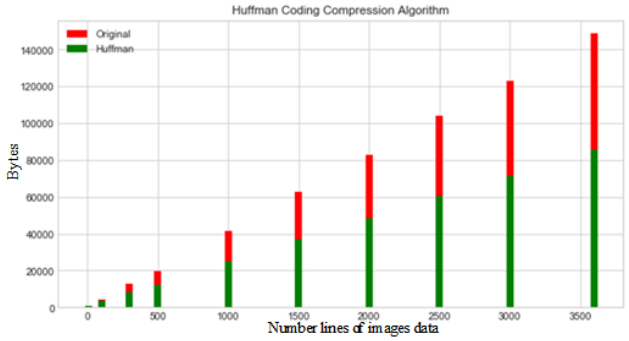

Fig. 3. Graph (bar) of results after testing the Huffman algorithm

The graphical representation of the data after compression in Figure 7 is done through dots. Whereas in Figure 3, the graphical representation is done through columns (the red part of the column is the reduced part of the bytes after compression).

Figures 4 and 5 show the results of the LZ77 compression algorithm. Figure 4 shows the graphic representation through the dots, and Figure 5 shows the graphical representation through the columns (the red part of the column represents the reduced part of the bytes after compression).

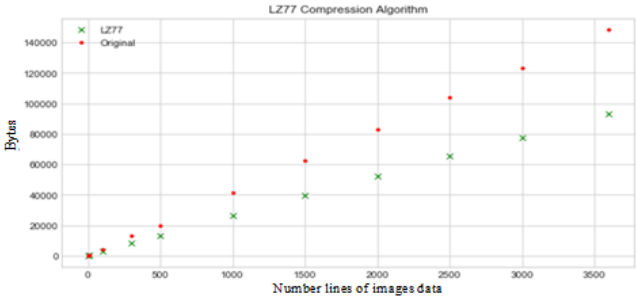

Fig. 4. Graph (scatter) of results after testing the algorithm LZ77

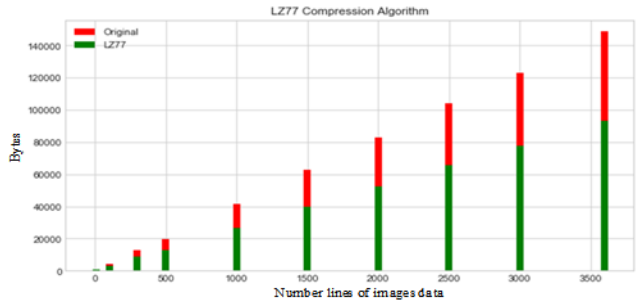

Fig. 5. Graph (bar) of results after testing the algorithm LZ77

The results obtained after testing the Run-Length Encoding algorithm can be seen graphically in Figure 6 and Figure 7.

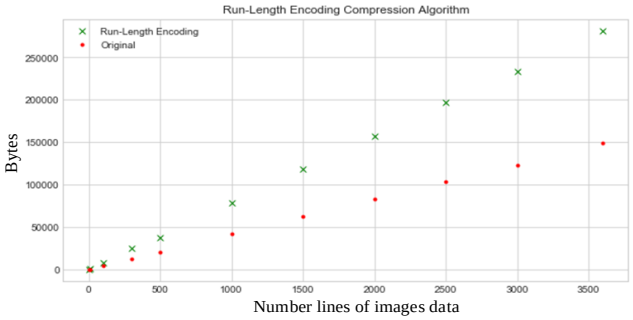

Fig. 6. Graph (scatter) of results after testing the algorithm Run-Length Encoding

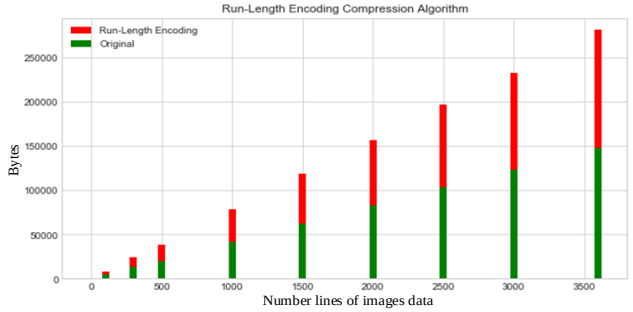

Fig. 7. Graph (bar) of results after testing the algorithm Run-Length Encoding

Figure 6 shows the graphic representation of results through the dots, and Figure 7, shows the graphical representation of results through the columns (the red part of the column represents the reduced part of the bytes after compression). The results are presented together in a single graph to make it easier to compare the results achieved between the three algorithms. Figure 8 shows a summary of the results of testing the three compression algorithms. From Figure 8, we see that slightly better compression results are achieved with the application of the Huffman algorithm than with the application of the LZ77 algorithm. Poor compression results are achieved by applying the Run-Length Encoding algorithm to this type of data.

From Figure 8, we can see that the Hoffman Algorithm offers a reduction of the space occupied by the original data by about 42.19%. The execution time for 3600 rows of image data is approximately 90.31 milliseconds or 0.0931 seconds. The LZ77 compression algorithm from testing shows that it offers a reduction of the space occupied by the original data by about 37.17%. The execution time for 3600 lines of the image data for this algorithm is about 65454.495 milliseconds or 65.45 seconds. The Hoffman algorithm provides 5.02% greater data compression than the LZ77 compression algorithm. The greater efficiency of the Hoffman Algorithm compared to the LZ77 compression algorithm is seen in terms of the time needed to perform compression, where the Hoffman Algorithm has an efficiency of 65.36 seconds better than the LZ77 compression algorithm.

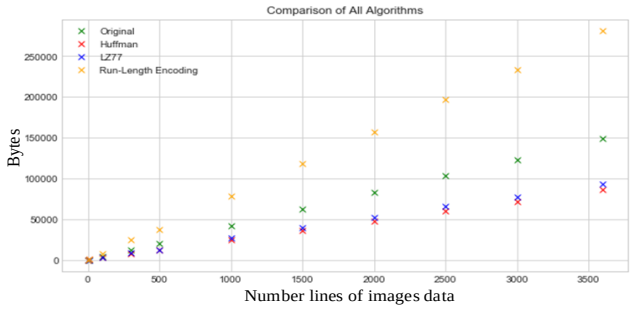

Fig. 8. Graph (scatter) comparing the original data with the image data after compression with Huffman, LZ77, and Run-Length Encoding algorithms

However, unsatisfactory results were obtained from the Run-Length coding algorithm testing (see Figure 8). Unlike the Hoffman compression Algorithm and the LZ77 compression algorithm, the Run-Length coding algorithm for any data case does not provide data size compression. But on the contrary, it requires additional storage space for this data after their compression. In addition, this can lead to an increase in the demand for data storage capacity up to 100% (for example, in this case, compressing a row of data, the need for storage space after data compression is double). The execution time for 3600 lines of the image data for the Run-Length coding algorithm is about 253.32 milliseconds or 0.25 seconds.

Conclusions

This research presents our efforts to analyze the application of IoT technology along the green line for state security purposes, as well as the development and analysis of data compression algorithms generated by IoT devices. For a large amount of data generated by thousands of multimedia sensors located along the state green borderline, data compression is beneficial for reducing data storage capacity requirements, transmission link capacity, and energy spent by sensors. The size and volume of data collected by IoT devices are constantly increasing. Therefore, the use of suitable compression algorithms plays an important role. There are several compression algorithms, but we developed and analyzed the performance of the three most popular compression algorithms, such as Huffman, LZ77, and Run-Length Encoding. These three data compression algorithms were modified and adapted for our research purpose, whereas the results graphically through Python libraries and the interactive Jupyter Notebook platform were visualized. The results of these three algorithms were tested, analyzed, and compared among themselves. Whereas the comparison and analytical analysis of the algorithms, in tabular form, are performed according to the measurement parameters, as well as the results are presented graphically. From the obtained results, we can see that although the Huffman and LZ77 algorithms gave satisfactory results by compressing up to 42.19% of the data size, the Run-Length Encoding algorithm did not provide any acceptable compression. The Run-Length Encoding algorithm only affects the increase in the size of the space required for data storage. The results show that the Hoffman Algorithm provides 5.02% more data compression than the LZ77 compression algorithm and a data compression time efficiency of 65.36 seconds better than the LZ77 compression algorithm. Therefore, from this testing and analytical analysis of the performance of the algorithms, we can conclude that the choice of the algorithm for the compression of specific data has a vital role in achieving good results.

***Authors**: The first author is Asoc. Prof. Dr Astrit Hulaj, University of Business and Technology, E-mail: astrit.hulaj@uni-pr.edu; Second author is Prof. Ass. Dr Bahri Prebreza* corresponding author, E-mail: bahri.prebreza@uni-pr.edu; Third author is Asoc. Prof. Dr Xhevahir Bajrami, E-mail: xhevahir.bajrami@uni-pr.edu; the University of Prishtina, Street" Sunny Hill", nn, 10000, Prishtina.*

REFERENCES

[1] Hulaj A., Bytyçi E., Kadriu V., An Efficient Tasks Scheduling Algorithm for Drone Operations in the Indoor Environment, *International Journal of Online & Biomedical Engineering*, 18(2022), No. 11, 42–57.

[2] Mahmud A., Husin H.M., and Yusoff N.M., Analysis on Literature Review of Internet of Things Adoption Among the Consumer at the Individual Level, *Journal of Information Science Theory and Practice,* 10(2022), No. 2, 45-73.

[3] Chanchí G. E., Ospina M., Rodriguez B. L., IoT system for CO2 level monitoring and analysis in educational environments. *Przeglad Elektrotechniczny*. 01(2023), No. 140, 140-146.

[4] Owczarczak R., Indoor thermal energy harvesting for battery-free IoT building applications, *Przeglad Elektrotechniczny.* 3(2023), No. 132, 132-136.

[5] Hulaj A., Shehu A., and Bajrami X., APPLICATION OF WIRELESS MULTIMEDIA SENSOR NETWORKS FOR GREEN BORDERLINE SURVEILLANCE, *Annals of DAAAM & Proceedings,* 27(2016), No. 27, 0845-0853.

[6] Hulaj A., Likaj R., & Bajrami X. Internet of Things Application for Green Border Surveillance, Based on Edge Detection Techniques. *International Journal of Intelligent Systems and Applications in Engineering,* 11(2023), No. 2, 702-709.

[7] Moffat A., Huffman coding, ACM Computing Surveys (CSUR), 52(2019), No. 4, 1-35.

[8] Dhawale N., Implementation of Huffman algorithm and study for optimization, *In 2014 International Conference on Advances in Communication and Computing Technologies (ICACACT 2014)*, 1-6, 2014.

[9] Shi P., Li B., Thike H.P., Ding L., A knowledge-embedded lossless image compressing method for high-throughput corrosion experiment, *International Journal of Distributed Sensor Networks,* 14(2018), No. 1, 1550147717750374.

[10] Correa [D.J., Pinto R.S.A., Montez C., Lossy Data Compression for IoT Sensors: A Review, *Internet of Things*, 19(2022), 100516.

[11] Liu X., An P., Chen Y., Huang X, An improved lossless image compression algorithm based on Huffman coding, *Multimedia Tools and Applications*, 81(2022), No. 4, 4781-4795.