https://dx.doi.org/10.22161/ijaems.66.2 *ISSN: 2454-1311*

Comparison of Compression Algorithms in text data for Data Mining

Roya Mahmoudi*, Mansoureh Zare

Department of Computer and Information Technology Engineering, Sabzevar branch, Islamic Azad University, Sabzevar, Iran.
*Corresponding Author

Abstract**—** *Text data compression is a process of reducing text size by encrypting it. In our daily lives, we sometimes come to the point where the physical limitations of a text or data sent to work with it need to be addressed more quickly. Ways to create text compression should be adaptive and challenging. Compressing text data without losing the original context is important because it significantly reduces storage and communication costs. In this study, the focus is on different techniques of encoding text files and comparing them in data processing. Some efficient memory encryption programs are analyzed and executed in this work. These include: Shannon Fano coding, Hoffman coding, Hoffman comparative coding, LZW coding and Run-length coding. These analyzes show how these coding techniques work, how much compression for these coding techniques. Writing, the amount of memory required for each technique, a comparison between these techniques is possible to find out which technique is better in which conditions. Experiments have shown that Hoffman's comparative coding shows a higher compression ratio. In addition, the improved RLE encoding suggests a higher performance than the typical example in compressing data in text.*

Keywords**— data compression, data mining, encryption, text data**.

I. INTRODUCTION

In computer science and information theory, using encryption techniques encrypts and compresses data using fewer bits or symbols than the original representation. Data compression is useful because it helps reduce the consumption of expensive resources, such as hard disk space or transmission bandwidth [3]. In the present situation where we are faced with the problem of data abundance and data disruption, data mining and data refinement and compression are used for this purpose. However, the problem is that the pressure of large compression may cause data loss [2]. That data is one of the valuable assets of any organization that the loss of some information can lead to major problems in the organization's decisions and strategies. The purpose here is to present and implement some data compression algorithms efficiently and compare them to the methods used for data compression in data mining. Calculating some of the important compression factors for each of these algorithms, comparing different encryption techniques and improving the performance of different data compression techniques, and selecting a suitable encryption method for data mining is the textual data considered.

Data mining, which has become very popular in recent years, refers to the use of data analysis tools to discover valid patterns and relationships that have been unknown until now [13]. These tools may include statistical models, mathematical algorithms, and learning methods (Machine Learning Methods) that are automated based on experience gained through neural networks or decision-making trees. Improves. Data mining is not limited to data collection and management, but also involves the analysis of data and the refinement and compression of data. Applications that explore data by examining text or multimedia files for data warehouses that have become popular [10]. This type of warehouse provides a variety of data types from text, image to storage and processing for use in organizational strategies and better understanding of data as information. Their advantage over databases is the ability to refine data and compress data to preserve important data and delete trivial data [5]. Each database has its query and query language. A multi-year problem in data analysis is increasing the size of the data set. This process demonstrates the need for more efficient compression programs and also for performing analytical operations that work directly on compressed data. Efficient compression schemes can be designed based on the misuse of patterns

https://dx.doi.org/10.22161/ijaems.66.2 *ISSN: 2454-1311*

and intrinsic structures in the data [9]. Data rotation is one of these features that can significantly enhance compression.

Text compression is a process to reduce text size by encoding more information. It uses this number of bits and bytes to store data which is somewhat reduced [1]. It requires a hassle-free technique that will not lose any data when compressing a text file. This approach returns the text file to its original state [4]. Text compression and decompression techniques are intended for natural languages such as large data sets of high redundancy English and other data with a similar sequential structure such as the source code of a program. However, this method can be used for any data type to achieve some compression [11]. The importance of text compression involves reducing storage hardware, data transfer time, and bandwidth, which is the use of file Compression can lead to a significant reduction in the cost of a drive or solid-state drive and improve network bandwidth, cost savings [7].

Textual data of various sizes are used in English to apply compression and decompression techniques. Text compression reduces data storage space. By reducing or eliminating redundancies, data compression is done. The encryption technique uses fewer bits than the original version and eliminates unnecessary information in this case. This is a harmless way in which the number of bits and bytes is cut off to store information.

Previous investigations

There are various methods for compressing data that take about forty years to develop. Shannon Fano and Hoffman developed and developed compression algorithms in the late 40s [10]. In 1949, Shannon and Fano devised a method for compression using a systematic method of assigning code words based on the probability of blocks called Shannon Fano coding. Another method of compressing data was found in 1951 by Huffman, known as the Huffman algorithm. Since then, the Huffman algorithm has been used for compression.

It has been shown in a review [13], that text data compression using the Shannon-Fano algorithm performs the same function as the Huffman algorithm, when all characters are repeated in a string, and when the expression is short and only one character in Text repeats, has the same functionality. When input data that have long text and data text has a more combinatorial character in the string or word, the Huffman algorithm has a greater impact on compression.

In another review [2], we tried compression ratio, compression time and decompression time for RLE encryption algorithms, Huffman encryption algorithm, Shannon Fan algorithm, Adaptive Hoffman encryption algorithm, Arithmetic encryption algorithm, and Lempel Zev Wel encryption. Were compared using random text files. The results show that as the text file size increases, the compression time increases. Medium-value encryption was less time consuming than the other algorithms for both Huffman and Shannon-Fano encryption approaches. In the LZW algorithm, it only worked fine for small files. The compression time of the Huffman algorithm was comparable to the encoding size at the highest time. The decompression time of all algorithms was less than 500,000 milliseconds except for the Adaptive Huffman and LZW algorithms. The compression ratio was similar for small size files except for RLE encoding, but LZW encoding worked best.

In another study [9], a comparison was made between RLE, Huffman, LZW computational encoding, first LZW then Hoffman and finally RLE, on the random doc, txt, BMP, tiff, gif and jpg files. , Which studies showed that LZW and Hoffman give almost the same results when used for compressing text files.

In another study [4], different methods of data compression algorithms such as LZW, Hoffman, fixed-length code (FLC) and Hoffman after using fixed-length code (HFLC) on English text files in terms of size compression, The ratio, time (speed) and entropy were examined. The best algorithms on all LZW encryption compression scales, then Hoffman, Hoffman, after using a fixed-length code (HFLC) and a fixed-length code (FLC), with entropy were 4.719, 4.855, 5.014 and 6.889, respectively.

In a similar study [5] that analyzed Hoffmann's algorithm and compared it with other common compression techniques such as Arithmetic, LZW, and RLE based on their use in different programs and its benefits. This is very efficient coding, and in RLE coding, when the sequence of pixels with fewer bits is more frequent, it significantly reduces the file size. The LZW algorithm is mostly used in TIFF, GIF and text files, which is a fast, harmless algorithm that is very easy to use, while the Hoffmann algorithm is used in JPEG compression, which produces optimal and low-volume code but is relatively slow.

In another paper [7], Shannon Fano coding, Hoffman coding, Adaptive Huffman coding, RLE, arithmetic coding, LZ77, LZ78, and LZW were tested using the Calgary body. In statistical compression techniques, accounting coding has been more effective than other methods. In another entropy review, the English text file for coding Shannon Fano, Hoffmann coding, Run-length (RLE), Lempel-Ziv-Welch (LZW) coding was calculated. The compression ratio for encoding Shannon Fano and

https://dx.doi.org/10.22161/ijaems.66.2 *ISSN: 2454-1311*

Hoffman was almost the same, and the two algorithms can save 54.7% of space. The compression ratio of Lempel-Ziv-Welch algorithms is low compared to the Hoffmann and Shannon Fano algorithms, and it has been concluded that the Hoffmann encryption algorithm is the best result for text files.

In another study [12], data was first compressed by each code based on the length of the run, such as Golomb code, FDR code, EFDR, MFDR, SAFDR, and OLEL encryption, and then another compressed data with Hoffman code was compressed. The double compression using the Hoffman code was 50.8% compression ratio. Better results were obtained for the data set with incremental data.

In a study [1] on execution time, compression ratio and compression efficiency in a distributed customer-server environment using four compression algorithms: Hoffmann algorithm, Shannon Fano algorithm, LZW algorithm, and Run-length encryption algorithm Was analyzed. A customer's data is distributed across multiple processors/servers and subsequently compressed by servers in remote locations and sent to the customer. The Sigrid framework was used, and the results showed that the LZ algorithm achieves better efficiency/scalability and compression ratio, but is slower than other algorithms. Hoffman coding, LZW coding, LZW-based Hoffman coding were compared for multiple and single compression. This showed that Hoffman-based LZW encryption can compress data more than any other three, while in normal mode the LZW compression ratio is based on Hoffmann 4.41, where the maximum compression ratio is maximal in the case of compression. The LZW build is 4.17. Hoffman-based LZW compression is in some cases better than LZW compression.

II. METHOD

As a test, five compression and stress relief techniques were used: Shannon-Fano coding, Hoffman coding, Hoffman comparative coding, and Lempel Ziv Welch coding, and improved RLE and RLE coding. Different sizes of text files have been used as data sets, and the compression ratio has been measured to determine which compression method is best.

III. RESULTS

The compression ratio depends on the size of the output file. The more compressed the output file is, the lower the compression ratio. When the compression ratio exceeds 1, the output file size is larger than the original input file size. Here we can see that in most modified RLE encrypted input files the compression ratio is greater than 1, meaning that these algorithms expand the original files rather than compress them. RLE encoding works best for files with sequential duplicates, but typically duplicate data files do not. For this reason, the compression ratio is greater than 1. The results also show that Shannon Fano encoding is better for most inputs than Huffman encoding since for some input files the symbol code length in Huffman encoding is so large that it increases the output files over the input files. Becomes original. It can be said that Repeated Huffman encoding shows the best compression ratio for most files.

If the code is shorter, the algorithms have less memory. Now if we consider the average length of the code, we will see that for Runlength encoding, we have no value for the average code length because no encoding is created in Run length. But for other techniques, Shannon Fano encodes less code than Huffman coding and modified Run-length coding. Here, too, Huffman coding shows the best average code length for most files. Again, if the standard deviations are smaller, the algorithms occupy less memory. Here, too, for run-length coding, we have no value for standard deviation since no coding is created in Run-length coding. But for other techniques, Shannon Fano has less standard deviation than Huffman coding and modified Run-length coding. Again, Huffman coding provides the best standard deviation for most files.

From the above analysis, we can say that Repeated Huffman coding works best for most cases. It has a smaller compression ratio, average code length, and standard deviation. So, it seems the best algorithm among them. This ultimately results in less memory than others for the output file after the final lapse.

If extra runtime is provided for Hoffman's frequent programming, then Shannon Fano's programming at lower runtime would work quite well. RLE coding for files with consecutive duplicates can be very effective. Modified RLE coding can be very useful if the temporary file intermediary created after applying the Huffman encoding to the input file has 0 and 1 consecutive iterations. Finally, each compression technique has its pros and cons. It depends on the content of the input files on how much better compression can be achieved. Table 1 shows the four input files after compression and resizing algorithms.

https://dx.doi.org/10.22161/ijaems.66.2 *ISSN: 2454-1311*

Table 1- Comparison of compression algorithms

Input File (.txt)	Input File Size (KB)	Characteristics	Huffman Coding	Shannon Fano Coding	Run length Coding	Modified Runlength Coding
1	156	Output File Size	87.5 KB	3.89 KB	291 KB	163 KB
		Compression Ratio	0.58	0.04	1.96	1.09
		Avg. Code Length	12.38	6.35	-	9.83
		Standard Deviation	12.33	0.19	-	15.33
2	117	Output File Size	77.3 KB	91.8 KB	245 KB	164 KB
		Compression Ratio	0.54	0.83	1.96	1.39
		Avg. Code Length	8.95	6.13	-	9.27
		Standard Deviation	13.89	0.10	-	14.77
3	453	Output File Size	698 KB	339 KB	388 KB	581 KB
		Compression Ratio	1.87	0.87	0.91	1.41
		Avg. Code Length	26.99	6.49	-	28.12
		Standard Deviation	414.78	0.27	-	412.79
4	470	Output File Size	765 KB	387 KB	919 KB	477 KB
		Compression Ratio	1.79	0.79	1.91	1.01
		Avg. Code Length	34.09	6.36	-	34.16
		Standard Deviation	450.87	0.28	-	447.01

IV. CONCLUSION

Text compression is very important in data refinement in data mining. Because data is an important part of data mining textual data. Therefore, it requires a no-loss technique so that no efficient information is lost when compressing or compressing information. In the investigations performed in this study on the performance and computation and comparison of compression ratios, mean code length and standard deviation for Shannon Fano coding, Huffman coding, Huffman repeated coding, Run-length coding, and modified Run-length coding, in The issue of how to compress was examined by each one so that, at present, the most effective algorithm can be used based on the size of the input text file, the type of content, available memory, and execution time to get the best results. A new approach to data compression is called "Run-length Run Algorithm", which offers much better compression than the run-length algorithm.

REFERENCES

[1] P. Peter, S. Hoffmann, F. Nedwed, L. Hoeltgen, and J. Weickert, "Evaluating the true potential of diffusion-based inpainting in a compression context," *Signal Processing: Image Commun.*, No. 46,40–53 (2016).

[2] M. N. Huda, "Study on Huffman Coding," Graduate Thesis, 2004.

[3] The Canterbury Corpus [Online]. http://corpus.canterbury.ac.nz/resources/cantrbry.zip. Accessed 30 May (2018)

[4] Habib, A., Rahman, M.S.: Balancing decoding speed and memory usage for Huffman codes using quaternary tree. Appl. Inf. 4, 5 (2017). https://doi.org/10.1186/s40535-016-0032-z

[5] S.R. Kodituwakku and U. S. Amarasinghe, "Comparison of Lossless Data Compression Algorithms for Text Data," Indian Journal of Computer Science and Engineering, vol. I(4), 2007, pp. 416-426.

[6] C. Lamorahan, B. Pinontoan and N. Nainggolan, " Data Compression Using Shannon-Fano Algorithm," JdC, Vol. 2, No. 2, September, 2013, pp. 10-17.

[7] M. A. Khan, "Evaluation of Basic Data Compression Algorithms in a Distributed Environment," Journal of Basic & Applied Sciences, Vol. 8, 2012, pp. 362-365.

[8] D. Belazzougui, S.J. Puglisi, Range predecessor and Lempel–Ziv parsing, in: Proc. 27th Annual ACM–SIAM Symposium on Discrete Algorithms (SODA), 2016, pp. 2053–2071.

https://dx.doi.org/10.22161/ijaems.66.2 *ISSN: 2454-1311*

[9] P. Bille, M.B. Ettienne, I.L. Gørtz, H.W. Vildhøj, Time-space trade-offs for Lempel–Ziv compressed indexing, Theoret. Comput. Sci. 713 (2018) 66–77.

[10] D. Kempa, D. Kosolobov, LZ-End parsing in compressed space, in: Proc. 27th Data Compression Conference (DCC), 2017, pp. 350–359, Extended version in https://arxiv.org/pdf/1611.01769.pdf.

[11] G. Navarro, A self-index on Block Trees, in: Proc. 24th International Symposium on String Processing and Information Retrieval (SPIRE), in: LNCS, vol. 10508, 2017, pp. 278–289.

[12] T. Nishimoto, T.I.S. Inenaga, H. Bannai, M. Takeda, Dynamic index and LZ factorization in compressed space, in: Proc. Prague Stringology Conference (PSC), 2016, pp. 158–170. [44] T. Nishimoto, T.I.S. Inenaga

[13] D. Belazzougui, S.J. Puglisi, Y. Tabei, Access, rank, select in grammar-compressed strings, in: Proc. 23rd Annual European Symposium on Algorithms (ESA), in: LNCS, vol. 9294, 2015, pp. 142–154.

[14] D. Kempa, N. Prezza, At the roots of dictionary compression: string attractors, in: Proc. 50th Annual ACM SIGACT Symposium on Theory of Computing (STOC), 2018, pp. 827–840.