

Enhancing Text Compression Method Using Information Source Indexing

Ahmed Musa[1*] and Ayman Al-Dmour

1. Department of Computer Engineering, Taif University, Taif-KSA. Sabbatical Leave: Al-Hussein Bin Talal University
2. Department of Computer Information Systems, Al-Hussein Bin Talal University

* E-mail of the corresponding author: as.shorman@gmail.com

Abstract

Text compression methods where the original texts are directly mapped into binary domain are attractive to compress English text files. This paper proposes an intermediate mapping scheme in which the original English text is transformed firstly to decimal domain and then to binary domain. Each two-decimal-digit value in the resulting intermediate decimal file represents the index to the location of each alphabet found in the original text. If the already indexed alphabet is seen again, it will be replaced by the previously given decimal-index number. The decimal file is converted into binary domain by assigning each decimal digit a 4-bit weighted code in according to its frequency of occurrence that is akin to BCD code. The assigned codes aim at generating an equivalent binary file with entropy as close as much to that of the original one. Thereafter, any conventional compression algorithm such as Lempel-Ziv algorithms can be applied to the generated binary file. The obtained compression ratios outperform those ones obtained when applying the same compression algorithm to the binary files generated either via direct mapping of the original text or via mapping the decimal file using Binary Coded Decimal (BCD) codes.

Keywords: Lossless data compression; Source encoding, LZW coding, Hamming weights, Compression ratio.

1. Introduction

Nowadays, the world witnesses a huge demand on storing or transmitting large volumes of data that are available in electronic form. This massive amount of data not only needs high capacity storage devices, but also requires long time to be transferred over the network. To save storage space and transfer time, intensive works and researches have been conducted to exploit the redundancy available in a given data. This redundancy can be removed using modern variants of compression techniques (Huffman 1952, Ziv & Lempel 1977 and 1978, Langdon & Rissanen 1981, Ian *et al.* 1987)

Modern compression techniques can downsize the large electronic-form data to a third of its original volume (Witten *et al.* 1995). Furthermore, searching or querying a compressed database of gigabyte of data becomes faster (up to 8 times) than that of the original one (Turpin & Moffat 1997, Moura *et al.* 2000). There have also been intensive works to find efficient searching algorithms within files compressed by Lempel-Ziv (LZ) variants (Navarro & Raffinot 1999, Navarro & Tarhio 2000). Thus, text compression becomes a key technology for text mining (Witten *et al.* 1999). Recently, devised compression techniques look to improve the compression ratios (CRs) as well as compression/decompression speed.

In this paper, we propose an enhanced English text compression method based on mapping the original text file into an intermediate Decimal File (DF). In the proposed method, an advanced processing on the intermediate file will be performed to count the frequency of occurrences of each decimal digit. According to this statistical distribution, a 4-bit binary weighted code is given to each decimal digit. This will generate a binary file that consists of two symbols (zero and one). Thereafter, any compression algorithm proposed in the literature can apply to nth-order extended binary file ((Musa *et al.* 2010, Elabdalla & Irshid 2001) instead of the original text file (Welch 1984) or intermediate DF. The obtained results show that there is an enhancement in the compression/decompression performance.

There are many compression algorithms devised in the literature. The question arises at this point: What is the most suitable algorithm that suits our work? The answer is to select one of lossless dictionary-based compression algorithms because of their relevance to our work (Powell 2014). Most practical dictionary compression techniques are built based on Lempel-Ziv algorithms and their variants (Ziv & Lempel 1977 and 1978, Powell 2014, Zeeh 2014). Amongst LZ variants, Lempel-Ziv-Welch (LZW) algorithm and its modified versions are the most popular (Welch 1984, Nelson 1989). Henceforth, LZW algorithm has been incorporated in the proposed

compression method.

2. Overview of the compression model

The key part of the compression method resides in constructing a model that enhanced its performance. Figure 1 shows the main aspects of the proposed compression model. The function of each aspect and the steps that the original text undergoes to yield its counterpart compressed version are elucidated in the following subsections.

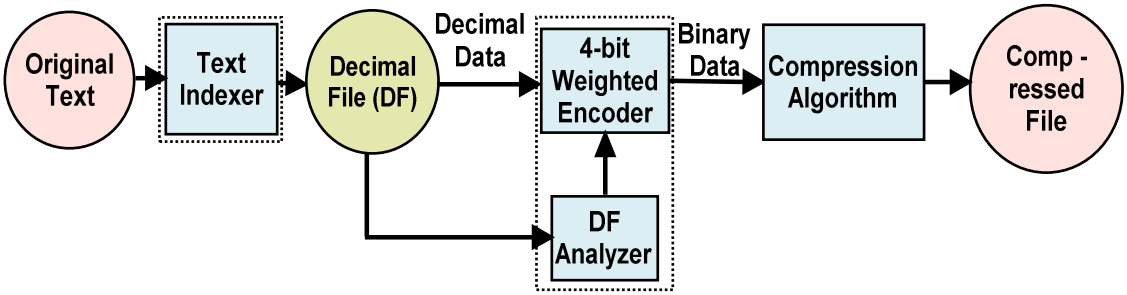


Figure 1: Compression process model.

2.1 Text Indexer and Decimal File analyzer

The first step in the proposed model is the indexing or tagging scheme using Text Indexer. This indexing process is mainly a replacement of each alphabet with a two-digit decimal number according to its location in the original text. If the already indexed alphabet is seen again, it will be assigned the previously given decimal-index.

This Text Indexer starts by receiving stream of alphabets of original text, see Figure 1. The indexer gives a 00 index or tag to the first alphabet appears in the original file. The second and third alphabets are replaced with 01 and 02 indices, respectively, and so on. Figure 2 illustrates the Text Indexer function when the input text is "Hello, World". The indexing scheme will continue till the end of the file. As a result, a DF consists of ten decimal digits that range from '0' to '9' is produced. One can notice that this indexing scheme will significantly reduce the number of alphabets available in the original English text from ninety-six to ten.

Text	"	H	e	l	l	o	,		w	o	r	l	d	"
Index	01	02	03	04	04	05	06	07	08	05	09	04	10	01

Digit	No. of occurrences	Assigned code
0	14	1111
1	3	1110
⋮	⋮	⋮
9	1	1100

Figure 2: Text indexer functionality.

The following step in the proposed model is to convert the intermediate DF into a binary one. This can be accomplished directly by replacing each decimal digit in DF by its corresponding BCD code. However, assigning BCD codes to decimal digits will not optimize the CRs of the compression algorithm. Hence, to attain better CRs, a code similar to BCD one is given to each decimal digit in according to its number of occurrences. The DF analyzer, shown in Figure 1, counts the number of occurrences of each digit and yields a statistical table that holds the available digits along with their number of occurrences sorted from largest to lowest. This table will be fed into the 4-bit weighted binary encoder.

2.2 The 4-bit weighted encoder

Once the aforementioned statistical table is readily available, the 4-bit weighted encoder starts assigning a 4-bit weighted binary code to each decimal digit available in the resulting DF. The question arises at this juncture is: How the 4-bit weighted encoder is constructed?

The 4-bit weighted encoder assigns binary codeword to each decimal digit as follows: The decimal digit that has the largest frequency of occurrence in the produced intermediate DF is assigned a code with a Hamming weight of four (i.e., 1111). In the same way, the next four most probable decimal digits are given 4-bit codewords that have a Hamming weight of 3. The rest five decimal digits are given 4-bit codewords that have a Hamming weight of 2. Figure 2 illustrates also the functionality of the 4-bit weighted encoder. Constructing the 4-bit weighted encoder in such away yields an equivalent binary file that has large number of symbol one and less number of symbol zero (Musa *et al.* 2010, Elabdalla & Irshid 2001). This will create bias towards increasing the number of one symbol on the account of another and thus optimize the difference between the entropy of the generated binary file multiplied by the code length of 8 and the entropy of the original text file. Now, one can apply LZW to the generated binary file on a bitwise basis.

3. Decimal-wise and Bit-wise LZW compression algorithm

The last step in the compression process is to apply LZW compression algorithm to manipulate the resulting intermediate DF on a decimal-wise basis or the generated binary file on a bitwise basis. It addition, the same compression algorithm can apply directly to the original text on a byte-wise basis.

Originally, LZW works on a byte wise or character wise basis. It starts out by prefilling the dictionary with 256 entries that are all possible 8-bit ASCII characters (ASCII 2008, Reynar *et al.* 1999). Afterwards, LZW starts reading data 8-bit (or one character) at a time and builds the dictionary by adding new substrings to it. The new substring is composed of the already present string in the dictionary concatenated with a new character.

Usually, to adapt to the memory limitations, the root of the newly created substring is replaced by a pointer, $P = 2^L - 1$, where L is an integer number. Thus, placing a limit on the value of P will lead to a limitation on the maximum number of entries that the dictionary can hold. For instance, if L is equal to 12-bit, the dictionary will hold approximately 4096 entries. Suppose LZW is applied directly to the original text with L equals to 12, each entry in the dictionary will be represented by 20 bit fixed-length codeword (*https://www.cs.duke.edu/csed/curious/compression/lzw.html*).

In this paper, LZW is applied to binary data to take the advantage of the repeated patterns and thus achieve compression. Hence, a slight refinement is performed in order to enable LZW to manipulate the nth-order extension binary source on a bitwise basis. In addition, some key parameters such as the extension-order, n, and the pointer, P, are to be passed to LZW. Once the extension-order, n, is defined, the LZW dictionary is prefilled with all 2n possible binary patterns. In fact manipulating source on a bitwise basis will ease the hardware implementation of the compression algorithm (Salomon 2007, Bell *et al.* 1990).

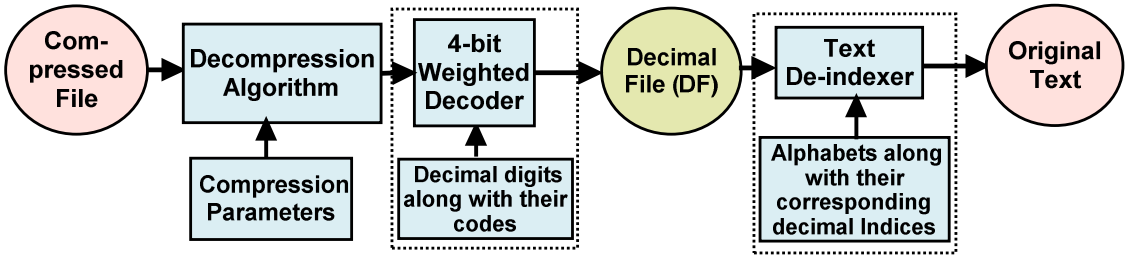
As an example, one can define n to be equal to 2. The first 4 entries in the dictionary are 00, 01, 10, 11. In turn, all new substrings available in the input binary file are built based on these entries. To cope with the limitation on the original LZW dictionary length, if n is chosen to be 2, the substring is replaced by a number, P, that can be represented by 18-bit code length. This means that the dictionary will hold maximum number of different binary patterns (say 262144 entries).

Now, one might apply LZW to the Intermediate DF on a decimal-wise basis. LZW starts out by prefilling the dictionary with the first 10 entries that range from '0' to '9'. These entries represent all possible decimal digits. Consequently, all possible substrings available in the file are added into the dictionary. The new substring is mainly composed from previously matched string concatenated with a one decimal digit. It is worth mentioning that manipulating the generated binary file at 4-bit extension order is akin to manipulating the DF on a decimal-wise basis. To cope again with the limitation of LZW on memory usage, the substring is replaced by a number, P, where P can be represented by 16-bit code length (for instance, a dictionary of 65536 entries).

4. Decompression Model

To retrieve the original text file back, a header file stores a collection of information used during the compression process must be readily available. The header file consists of the following information: (i) the alphabets available in the original text along with their corresponding decimal indices, (ii) the available decimal digits

along with their 4-bit codes, and (iii) any other parameters used by the compression technique. This file is a key input to the entire decompression process which works exactly in an opposite way to the compressor one, see Figure 3.



Figures 3: Decompression process model.

5. Experimental Results

The effect of mapping the original text file into decimal domain and then converting the resulting file to a binary domain will be investigated in experiments. To conduct the experiments, we write a Java program that performs: (1) indexing the original text files and produce intermediate DF, (2) performing statistical analysis on the resulting intermediate DF to count the number of occurrences of each decimal digit, and (3) generating the binary file either via using BCD or 4-bit weighted codes.

The experiments are conducted on a test suit that has two different text files *WWEND.txt* and *MissLife.txt* (*http://www.gutenberg.org/browse/authors/t#a53*). These files proof to include wide range of English Text. Table 1 lists these files along with their sizes as well as their entropy computations. In addition, it shows the corresponding decimal files, sizes along with their entropy computations.

Table 1: Files of the test suite and their corresponding Decimal Files.

Original			Decimal		
File Name	Size (byte)	Entropy H(s) (bits/character)	File Name	Size (# decimal digits)	Entropy (bits/symbol)
WWEND.txt	1186944	4.473	*WWEND_Dec.txt*	2373888	2.525
MissLife.txt	839942	4.544	*MissLife_Dec.txt*	1679884	2.584

Each text file in the suit can be converted to binary domain in three different ways. The first way is to map the original text file directly into binary domain using dynamic text mapping scheme (Elabdalla & Irshid 2001). This way yields a binary file named B1. The second binary file, B2, is generated via replacing each decimal digit in the resulting intermediate DF with its corresponding BCD representation. The last way produces the binary file, B3, by assigning a 4-bit weighted code to each decimal digit in the intermediate DF, as was previously described. Table 2 shows the generated binary files (B1, B2, and B3) of each text in the test suit, the number of zeros and ones in each file, and their entropy computations.

Table 2: The corresponding binary files and their entropy computations

File Name	Mapping scheme	No. of ones	No. of Zeros	Entropy H(B)
WWEND_B1.txt	Direct & dynamic	7081197	2414355	0.818
WWEND_B2.txt	BCD	1809604	7685948	0.703
WWEND_B3.txt	4-bit weighted code	7822197	1673355	0.672
MissLife_B1.txt	Direct & dynamic	4983673	1735863	0.824
MissLife_B2.txt	BCD	1327110	5392426	0.717
MissLife_B3.txt	4-bit weighted code	5486667	1232869	0.687

Table 3 shows the entropy of original text along with its eight-order extended binary files entropy computations. In addition, the table shows the difference in entropy ΔH1, ΔH2, and ΔH3 between the original text source H(s) and the 8th-order extended generated binary sources H(B1), H(B2), and H(B3), respectively. From the figures listed in the table, one can observe that ΔH3 is always less than ΔH2 and ΔH1. Therefore, better CRs are expected to acquire when the source indexing or tagging scheme is utilized.

Table 3: The difference in entropy computations.

		8^{th}-Order extended binary source Entropy (bits/character)			Difference in Entropy		
File Name	Original Text Entropy H(S) (bits/Character)	8*H(B1)	8*H(B2)	8*H(B3)	ΔH1	ΔH2	ΔH3
WWEND.txt	4.473	6.544	5.621	5.374	2.071	1.148	0.901
MissLife.txt	4.544	6.594	5.735	5.501	2.049	1.191	0.957

To illustrate the effect the proposed indexing scheme on the performance of the compression method, LZW, we write another java program that will take the binary file as an input and yields the corresponding compressed file. Figure 4 displays the CRs (measured in bits per character) obtained when the resulting binary file of *wwend.txt* is manipulated using bitwise LZW at different extension orders, n. The results show that CRs are better when the original text file is mapped to binary one via the decimal domain. This is because the intermediate DF will have only ten alphabets instead of 96. Therefore, performing an advanced probability study on the DF is better than doing so on the original text file. In addition, mapping the intermediate DF to binary domain using 4-bit weighted codes will further enlarge the probability of symbol one on the account of the probability of symbol zero. This will lead to achieve better CRs.

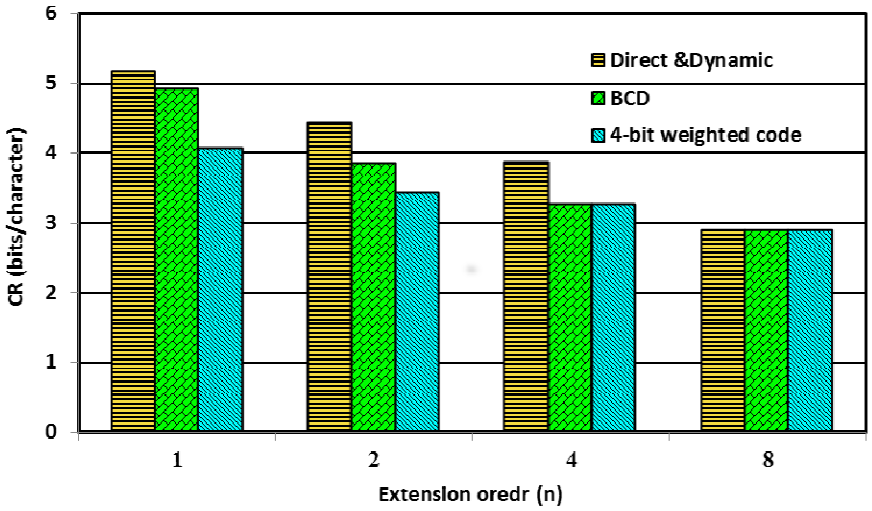


Figure 4: CRs using bitwise LZW for *wwend*.txt.

To verify the results and its consistency, the same experiment is conducted on another file listed in Table 1, *MissLife.txt*. The amount of CRs displayed in Figure 4 follows the same trend of *wwend.txt's* results shown in Figure 5.

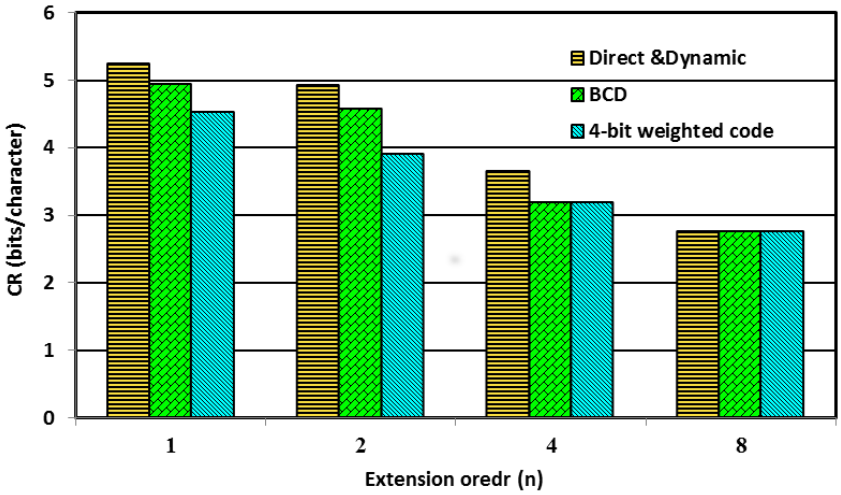


Figure 5: CRs using bitwise LZW for *MissLife.txt.*

One might apply LZW on a decimal-wise basis to the files *wwend_Dec.txt* and *MissLife_Dec.txt*. The obtained CRs measured in symbols per character are equal to 0.816 and 0.798, respectively. These values are equal to those ones obtained when manipulating *WWEND_B3.txt* and *MissLife_B3.txt* at an extension order of 4. . It is a matter of fact that each decimal digit can be represented by a 4-bit code. Thus, for instance, multiplying 0.814 by 4 is equal to 3.264. This is the value obtained when the corresponding binary file, *WWEND_B3.txt*, is manipulated at n equals to 4, as shown in Figure 2.

The results displayed in Figures 4 and 5 are obtained when LZW dictionary length can increase indefinitely. As was previously mentioned, the dictionary length is limited by the pointer value, P. To testify the influence of dictionary length on the compression/decompression process, bitwise LZW with a limit placed on P is applied to nth-bit extension order of the generated binary file. Figure 6 illustrates this influence when *WWEND_B3.txt* is manipulating. From this figure one can observe that the CRs are increased if the pointer values are decreased and reach the optimal when the dictionary has indefinite length. However, the increment in CRs is amortized by memory utilization and the less time needed to find the root of the subsequence in the specified dictionary.

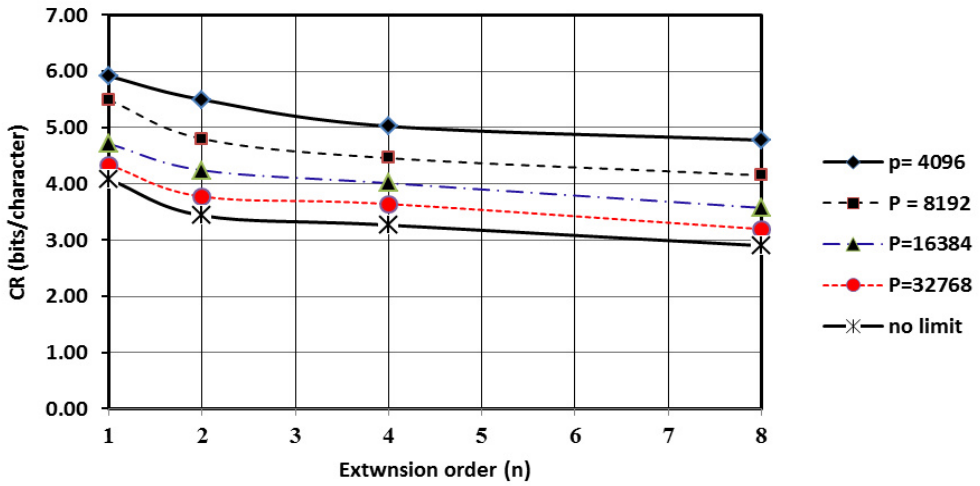


Figure 6: CRs using different pointer limits, P, of *WWEND_B3.txt.*

6. Conclusions

An enhanced compression method that indexes or tags the original text file using two-digit decimal number is presented. Incorporating this indexing scheme in the compression method improves the CRs. This is because the indexing process significantly reduces the number of alphabets available in the original text from ninety-six to ten. Thus, the correlation between decimal digits in the resulting DF will maximize.

Further, the decimal file can be converted into binary ones using BCD or the proposed 4-bit weighted codes. Applying the compression algorithm, LZW, on a bit-wise basis to the binary file generated using 4-bit weighted codes gives better CRs than applying the same technique to the binary file generated using the BCD codes. In addition, manipulating the binary files at an extension order of 4 gives the same CR when manipulating the resulting DF on a decimal-wise basis. The speed of the compression/decompression is still fair. This challenge opens the door for further research. To cope with the memory limitation, a limit on the number of entries that the dictionary can hold has been placed. This limitation will degrade the CRs. On the other hand, the compression/decompression speed is improved since the search time for the root of the newly created substring is significantly reduced.

References

Huffman, D. (1952), "A Method for the Construction of Minimum Redundancy Codes," in Proc. IRE, vol. 40, no. 9, pp. 1098-1101.

Ziv, J., and Lempel, A. (1977). A universal algorithm for sequential data compression. IEEE Trans. Information Theory, 23, 337-343.

Ziv, J., and Lempel, A. (1978). Compression of individual sequences via variable-rate coding. IEEE Trans. Information Theory, 24, 530-536.

Langdon, G., and Rissanen, J. (1981). Compression of black-white images with arithmetic coding. IEEE Trans. Communication, 29, 858-867.

Ian, H., Radford, M. & Cleary, G. (1987), "Arithmetic coding for data compression", Communications of the ACM, New York, Volume 30, Issue 6, pp: 520-540

Witten, I., Moffat, A. & Bell T. (1995). Compression and full-text indexing for digital libraries. In Lecture Notes in Computer Science Series (LNCS) 916, Digital Libraries, Current Issues, pages 181–201. Springer-Verlag.

Turpin, A. and Moffat, A. (1997), Fast file search using text compression, Proceedings of the 20th Australian Computer Science Conference, pp. 1–8.

Moura, E., Navarro, G., Ziviani, N. & Baeza-Yates, R. (2000), Fast and flexible word searching on compressed text, ACM Transactions on Information Systems 18(2), 113–139.

Navarro, G. and Raffinot M. (1999). A General Practical Approach to Pattern Matching over Ziv-Lempel Compressed Text. Proc. CPM'99, LNCS 1645. Pages 14-36.

Navarro, G. and Tarhio, J. (2000), “Boyer-Moore String Matching over Ziv-Lempel Compressed Text”, Proc. CPM'2000, LNCS 1848. Pages 166-180, 2000.

Witten, I. et al. (1999). Text mining: A new frontier for lossless compression". Proceedings of Data Compression Conference, Snowbird, Utah.

Powell, M. (2014), Evaluating lossless compression methods [Online]. Available: http://corpus.canterbury.ac.nz/research/evaluate.pdf, last accessed February 2014.

Zeeh, C. (2014), The Lempel Ziv algorithm [online]. Available: http://tuxtina.de/files/seminar/LempelZiv.pdf, last accessed February 2014.

Welch, T. (1984), "A Technique for High-Performance Data Compression," Computer 17, pp. 8-18, 1984.

Nelson. M. (1989), "LZW Data Compression," in Dr. Dobb's Journal, 1989, pp. 29-36, 86-87.

ASCII character code reference [Online]. Available: http://nemesis.lonestar.org/reference/telecome/codes/ascii.html, last accessed March 2008.

Musa, A., Al-Dmour, A., Al-Khaleel, O. & Irshid, M. (2010), “An efficient compression technique using Lempel-Ziv algorithm based on dynamic source encoding scheme,” International Journal of Information and Communication Technology (IJICT), Vol. 2, No. 3, Inderscience Enterprises, pp. 210-219, 2010.

Elabdalla, A., and Irshid, M. (2001). An efficient bitwise Huffman coding technique based on source mapping. Journal of Computers and Electrical Engineering, 27, pp. 265 – 272.

Reynar, J., Herz, F., Eisner J. & Ungar L. (1999), Lempel- Ziv data compression technique utilizing a dictionary pre-filled with frequent letter combinations, words and/or phrases, US Patent Issued on September 1999. Available: http://www.patentstorm.us/patents/5951623.html.

https://www.cs.duke.edu/csed/curious/compression/lzw.html , last accessed March 2014.

Salomon, D. 2007. Data Compression: The complete reference, 3rd Ed., Springer, Berlin-New York.

Bell, T., Cleary, J. & Witten, I. (2019), Text Compression, Prentice-Hall, Englewood Cliffs, New Jersey, 1990.

http://www.gutenberg.org/browse/authors/t#a53, last accessed February 2014.

Ahmed Musa was born on October 20, 1974 in Irbid, Jordan. He received a B.S. and M.S. in 1997 and 2000, respectively, in electrical and computer engineering from Jordan University of Science and Technology, Irbid, Jordan. He worked as a telecom engineer in Jordan Telecom from 1999-2001. In the fall of 2001, he joined the Ph.D. program in Computer Engineering at the University of Texas at El Paso. While his period at UTEP, Musa worked at the Fiber Optic modeling and simulation Laboratory as a Research Assistant from 2004 to 2006. He also was appointed as instructor at UTRP from 2003 to 2006. In 2006, Musa joined the faculty of Computer Engineering at Al-Hussein Bin Talal University (AHU), Ma’an – Jordan. In the fall of 2006, Musa is charged with leading the Department of Computer Engineering at AHU to the next level of distinction and assisting the college of Computer Engineering and Information Technology in promoting excellence in research and teaching. In 2007, Musa received the AHU Award for his great academic achievement and his research. In fall 2011, Musa was promoted to an associate professor rank. Currently, he is on unpaid leave from Department of Computer Engineering, AHU and works at Department of Computer Engineering, Taif University, Kingdom of Saudi Arabia.

Ayman Al-Dmour was born in 1970 in Adnaneih-Karak. He received his B.Sc in Electronic/Communication Engineering in 1994 from Jordan University of Science and Technology, Irbid-Jordan. He perused his M.Sc and Ph.D. in 2003 and 2006, respectively, both in Computer Information Systems from Arab Academy, Amman, Jordan. His research interests are in image processing and data compression. In 2011, he was appointed as an acting dean, faculty of information technology, Al-Hussein Bin Talal University (AHU). At AHU he has lead the department of Computer Information Systems and the Computer and Information Technology Center. Previously, he worked with Orange Jo as a telecom engineer. In 2012, He took a sabbatical leave from AHU and joined the department of Information Technology, Taif University, Taif – Kingdom of Saudi Arabia.

The IISTE is a pioneer in the Open-Access hosting service and academic event management. The aim of the firm is Accelerating Global Knowledge Sharing.

More information about the firm can be found on the homepage:
http://www.iiste.org

CALL FOR JOURNAL PAPERS

There are more than 30 peer-reviewed academic journals hosted under the hosting platform.

Prospective authors of journals can find the submission instruction on the following page: http://www.iiste.org/journals/ All the journals articles are available online to the readers all over the world without financial, legal, or technical barriers other than those inseparable from gaining access to the internet itself. Paper version of the journals is also available upon request of readers and authors.

MORE RESOURCES

Book publication information: http://www.iiste.org/book/

IISTE Knowledge Sharing Partners

EBSCO, Index Copernicus, Ulrich's Periodicals Directory, JournalTOCS, PKP Open Archives Harvester, Bielefeld Academic Search Engine, Elektronische Zeitschriftenbibliothek EZB, Open J-Gate, OCLC WorldCat, Universe Digtial Library , NewJour, Google Scholar