http://ansinet.com/itj

ITJ

ISSN 1812-5638

INFORMATION TECHNOLOGY JOURNAL

ANSInet
Asian Network for Scientific Information
308 Lasani Town, Sargodha Road, Faisalabad - Pakistan

Information Technology Journal 9 (4): 673-679, 2010
ISSN 1812-5638

A Dynamic and Secure Arabic Text Compression Technique Using Bitwise Lempel-Ziv Algorithm

[1]A. Musa, [2]A. Al-Dmour, [3]F. Fraij, [4]O. Al-Khaleel and [5]M. Irshid
[1]Department of Computer Engineering,
[2]Department of Computer Information Systems,
[3]Department of Software Engineering, Al-Hussein Bin Talal University, P.O. Box 20, Ma'an 71111, Jordan
[4]Department of Computer Engineering,
[5]Department of Electrical Engineering, Jordan University of science and Technology, Irbid 22110, Jordan

Abstract: This study presented a new Arabic text compression technique in which an input file is preprocessed to produce a new binary file. The preprocessing maps a codeword to each character based on its occurrences in the input file. The binary file is then compressed using a bit-wise Lempel-Ziv algorithm. The new technique has been implemented and examined using different files. The results showed that the new technique, using extension-order of 8 or 4, achieves a compression ratio with the range of 4.25 to 4.7 bits/character, respectively. The improvement in compression efficiency is due to the significant reduction in the number of symbols in the generated binary file. It is noteworthy that embedded security on the compressed file is naturally acquired due to the automatic generation of the private key via mapping.

Key words: Character mapping scheme, compression ratio, source encoding, source entropy, hamming weight

INTRODUCTION

The advancements in information and communication technologies show a continuous demand to exchange significant volumes of data. As a result, improving current data compression techniques to drastically reduce such volumes of exchanged data is indispensable. It is also an advantage to naturally place security on the data exchanged without introducing extra overhead.

The efficiency of any compression technique can be measured based on four metrics. These metrics are (1) compression and de-compression speed, (2) the amount of memory size that is consumed in the compression and de-compression processes, (3) the ease of implementing the technique and (4) the compression ratio, measured in bits/character. Many researchers in data compression aimed at improving the aforementioned metrics.

In the early days, compression algorithms presented in the literature tended to use small amount of memory and CPU time (Elabdalla and Irshid, 2001; Bell *et al*., 1990). Recently and due to the advancement in VLSI technology both of these metrics become cheaper. Therefore, nowadays, compression algorithms have centered their attention on achieving better compression ratios.

The Lempel Ziv algorithm and its variants, is an algorithm for fixed-length, lossless data compression. Originally, the two algorithms proposed by Jacob Ziv and Abraham Lempel in their landmark papers in 1977 and 1978. Lempel Ziv algorithms are used in variety of applications such as gzip, GIF image compression and the V.42 modem standard (Sayood, 2000; Zeeh, 2009). The LZ algorithms can be used to compress plain text files (Held and Marshall, 1991; Kida *et al*., 1999). Naturally, plain texts may be composed of characters or binary data. According to this composition, these algorithms are described as character-wise or bit-wise, respectively.

Prior to applying LZ algorithm, the original text can be mapped into a binary equivalent counterpart that produces sequences of 0 and 1 sec. This mapping aims at improving the compression ration of the algorithms and facilitating their hardware implementation. Mapping is performed in two different ways either statically (Musa and Irshid, 2008) or dynamically (Musa *et al*., 2008). Static mapping utilizes a predetermined character-to-ASCII table for all input files, i.e., the table is file-independent. The dynamic mapping, however, constructs such a table on-the-fly for each input file, i.e., the table is file-dependent.

In dynamic mapping, the codeword assignment for each character is determined based on the statistical

Corresponding Author: Ahmed Musa, Department of Computer Engineering, Al-Hussein Bin Talal University, P.O. Box 20, Ma'an 71111, Jordan Tel: +962 (3) 217-9000 Fax: +962 (3) 217-050

analysis of an input text file. The analysis of the frequency of English characters has been investigated (Bell *et al*., 1990; Jaradat and Irshid, 2001; Powell, 2009). However, finding such analysis for Arabic letters is challenging (intellaren). Albeit Arabic text compression has been investigated (Jaradat *et al*., 2006a; Kanaan *et al*., 2006), preprocessing the input Arabic Text Files (ATFs) and manipulating them on bit-wise level have not been introduced. This study introduces a new compression technique to minimize the compression ratio of AT using a bit-wise Lempel-Ziv (BWLZ) algorithm. Moreover, this technique uses dynamic source mapping scheme rather than the static one. This will provide a private key tat will be used to place a high security on the ATF.

CHARACTER MAPPING SCHEMES

The abstraction model of the proposed Arabic compression/de-compression technique based on the dynamic mapping scheme is shown in Fig. 1. Figure 1 depicts that a statistical analysis of ATFs has performed using AT analyzer. In the light of this analysis, each character available in the ATF is mapped into a fixed length Hamming weighted binary codeword. Thus, the AT encoder will assign a weighted codeword for each character.

Mapping ATF can be accomplished using two different schemes ASCII and dynamic. In the ASCII scheme, the mapping table is constructed based on the ASCII code of each Arabic character (Wikipedia). Thus, the time used to construct such a table is saved as there is no need to perform a pre statistical analysis on the ATF. On the other hand, dynamic mapping scheme is similar to the ASCII one except that different mapping table is constructed that depends on the statistics of the input file. Hence, the dynamic scheme has the disadvantage of imposing extra time to dynamically construct the mapping table.

Although, ASCII mapping scheme reduces the overall time complexity of compression algorithms, dynamic character mapping scheme is utilized in this study. The reason for this is that the compression ratio will be decreased and therefore, the compression speed will be higher.

The corner stone in the analysis stage of ATF resides in obtaining the mapping table. It is difficult to find typical files, especially for Arabic language, that can be used to evaluate the performance of compression methods. For this reason, in this study, we randomly select two famous Arabic text files, تحفة العروس.txt and الرحيق المختوم.txt, to scrutinize the proposed algorithm (intellaren). For the dynamic character mapping, each source will be processed and a corresponding statistical mapping table will result. Table 1 and 2 show the frequency of occurrence of some Arabic characters available in these files ordered in a descending order.

Table 1: Probability of some Arabic characters exist in الرحيق المختوم.txt along with its corresponding ASCII Code and dynamic assigned code

Characters	Characters pronunciation/name	Probability	ASCII code	Assigned code
•	Space	0.1759	00100000	11111111
ا	Alef	0.0919	11000111	11111110
ل	Lam	0.0912	11100001	11111101
م	Meem	0.0478	11100011	11111011
و	Waw	0.0449	11100110	11110111
…..		…..	…..	…..
ب	Beh	0.0302	11001000	11111100
…..		…..	…..	…..
ع	Ain	0.0255	11100110	11110110
ت	The	0.0217	11001010	11101110
ف	Feh	0.0210	11011100	11011110
…..		…..	…..	…..
CR	Carriage return	0.0042	00001010	11001110
…..		…..	…..	…..
؟	Arabic question mark	0.0007	00001010	11011001
…..		…..	…..	…..
*	Asterisk	0.0003	00101010	10111001
!	Exclamation mark	0.000002	00100001	11100011
…..		…..	…..	…..

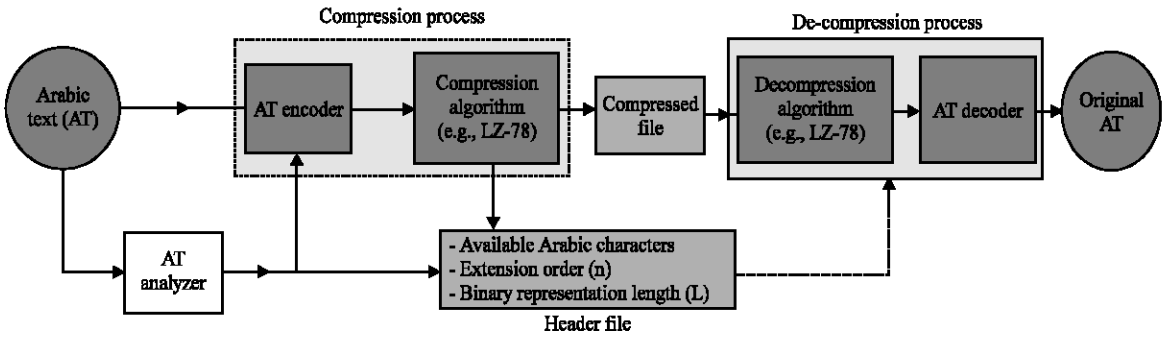

Fig. 1: The abstract model of compression and de-compression process based on mapping scheme

Table 2: Probability of some Arabic characters exist in تحفة العروس.txt along with its corresponding ASCII code and dynamic assigned code

Characters	Characters pronunciation/name	Probability	ASCII code	Assigned code
•	Space	0.187	00100000	11111111
ا	Alef	0.094	11000111	11111110
ل	Lam	0.079	11100001	11111101
و	Waw	0.041	11100110	11111011
م	Meem	0.039	11100011	11110111
.....	.....	.....	.....	.....
ر	Reh	0.028	11001110	11111100
ب	Beh	0.024	11001000	11111010
.....	.....	.....	.....	.....
ت	The	0.024	11001010	11110110
ع	Ain	0.023	11100110	11101110
ف	Feh	0.020	11011100	11011110
.....	.....	.....	.....	.....
CR	Carriage return	0.0062	00001010	10101111
.....	.....	.....	.....	.....
؟	Arabic question mark	0.0005	00001010	11011001
.....	.....	.....	.....	.....
*	Asterisk	0.000002	00101010	10011101
؛	Arabic semicolon	0.000003	10111010	11100011
.....	.....	.....	.....	.....

The mapping process is conducted using the following rule: the character in the ATF with the largest frequency of occurrence is replaced by a codeword with the largest Hamming weight. For instance, the codeword of a Hamming weight of eight (i.e., all ones eight-bit binary codeword) is assigned to the Arabic character that has the largest frequency of occurrence in the given text. The next most probable character in the ATF is assigned a codeword with the Hamming weight equal to 7. This assignment is repeated until the character with the rare frequency of occurrence is assigned a codeword with a minimum Hamming weight. In general, codewords with a Hamming weight of (N-k) are assigned to the characters as follows:

$$\binom{N}{k} = \frac{N!\ (N-k)!}{k!} \tag{1}$$

where, N is the length of the assigned binary codeword and k is ranged from 0 to N.

Table 1 and 2 show the ASCII code of each Arabic character along with the dynamic one. One can observe that in the dynamic mapping scheme the code assigned to each Arabic character varies from one file to another. For instance, the code assigned to the ع (Ain) character in الرحيق المختوم.txt file is 11110110 and in تحفة العروس.txt is 11101110. This variation in the code assignment is due to the difference in the frequency of occurrence of this character in the input files.

THE ENTROPY OF ARABIC TEXT FILE

The first step in constructing the mapping table is to statistically analyze the input ATF and count the frequency of occurrence of each character in it. Based on this statistical analysis the entropy of the original ATF is given by:

$$H(s) = -\sum_{i=0}^{M} p_i \log_2 p_i \tag{2}$$

where, M is the number of characters available in the ATF and p_i is the probability of occurrence of each one of them.

According to the statistical analysis of ATF, available characters are sorted in descending order and stored in a header file (Fig. 1). In addition, each character is mapped into a unique fixed-length weighted binary code. Thus, an equivalent binary source that consists of two symbols, zero and one, has obtained. The entropy of the resulting binary source is:

$$H(B) = -[p_0 \log_2 p_0 + p_1 \log_2 p_1] \tag{3}$$

where, p_0 and p_1=1- p_0 are the probability of symbols zero and one in the resulting binary file, respectively.

Assigning a codeword for each character using the aforementioned rule will lead to a dynamic AT encoding scheme. As a result, the difference between the first-order entropy of the resulting binary source, H(B) given in Eq. 1, multiplied by the length of the code, N and the entropy of the original source, H(S) given in Eq. 2, will be optimal. This means that the difference in entropy, ΔH, between the original source and the nth-order extended binary source, is less when the dynamic mapping scheme is used:

$$\Delta H = H(S) - N \times H(B) \tag{4}$$

Now one can use Eq. 2 and 3 to compute the entropy of الرحيق المختوم.txt and تحفة العروس.txt files. Table 3 shows the entropy computation of these files. From the values shown in these tables one can observe that the difference in entropy between the original source and the eighth-order extended binary one is less when the dynamic mapping scheme is used. A further study on reducing the entropy of text files based on file splitting was performed by Jaradat *et al.* (2006b). This way of splitting files can be also utilized in this study to improve the compression technique metrics.

Table 3: Summary of entropy computation based on the first-order binary source for tested files

File name	Source entropy H(S) (bits/character)	Entropy (bits/symbol) ASCII mapping	Entropy (bits/symbol) Dynamic mapping
الرحيق المختوم.txt	4.41	7.98	5.03
تحفة العروس.txt	4.86	7.92	5.14

BITWISE LEMPEL-ZIV ALGORITHM

Here, the bitwise Lempel-Ziv (BWLZ) algorithms are explained by a simple example. The example utilizes a version of LZ algorithms, namely LZ-78, to manipulate the nth-order extension of a binary source. Figure 2 graphically shows the example where the extension-order is 2. It is worth mentioning that this example is for illustration purposes and does not show the advantages of the algorithm. The advantages of the algorithm become clear as the size of the input stream increases.

The algorithm, in the example, manipulates a binary input stream, 11110011111111. The upcoming subsections will illustrate in-depth the operation of LZ-78 compression algorithm and de-compression one on this input stream. The compression side of the algorithm condenses the stream. Afterwards, the recipient receives the condensed stream and the de-compression restores the original data.

Bitwise compression process: The compression process starts by constructing a compression dictionary; a table consists of two columns, namely an index and the binary subsequence. This table is used as an intermediate step to obtain the condensed version of the input stream. The dictionary is pre-filled by initialization subsequences (Reynar *et al*., 1999). The number of initialization subsequences depends on the extension-order (n).

In the aforementioned example, as the extension-order of 2 is used, the first four rows in the dictionary are filled with 00, 01, 10 and 11, respectively. The algorithm reads the input stream in blocks of 2 bits. It checks whether the read pattern has been previously seen in the dictionary. If so, it proceeds and reads another block and concatenates it with the one in hand and then performs the check process on the new pattern once again. Otherwise, the algorithm adds another binary subsequence in the next free row in the table. For instance, in Fig. 2, the algorithm reads the first 2 bits, namely 11, by checking the dictionary; the pattern 11 has been already stored at the index 4. Therefore, the algorithm proceeds and reads another 2 bits and concatenates it with the one in hand. Therefore, the

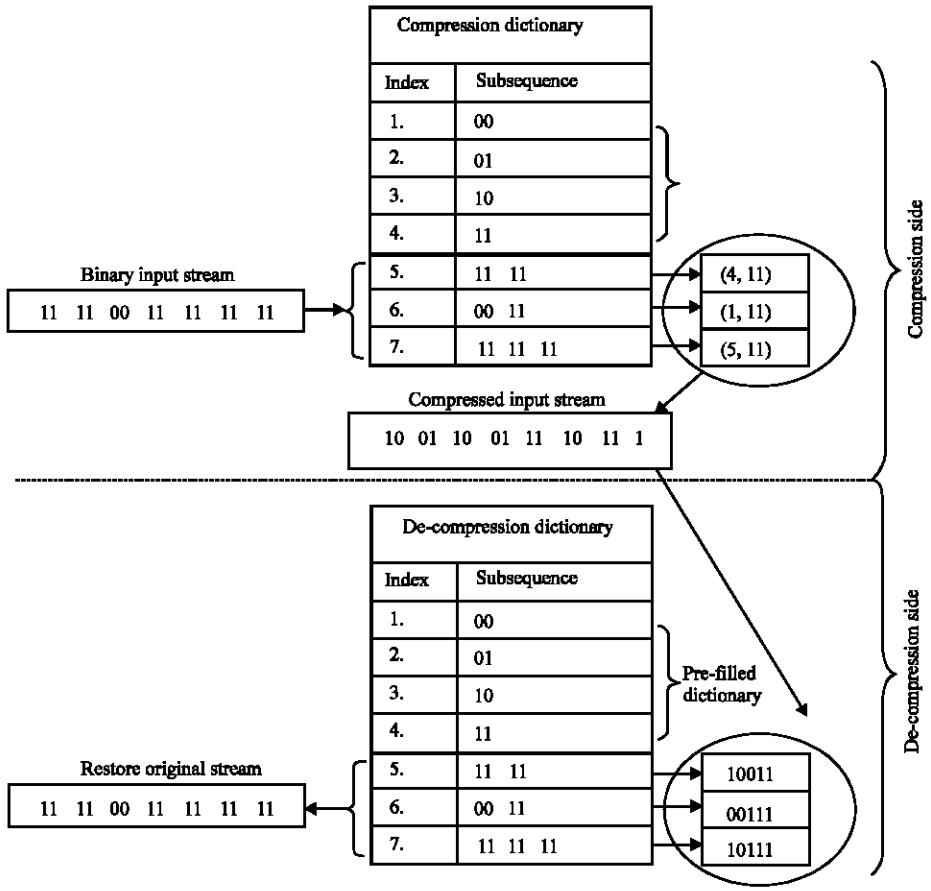

Fig. 2: Example of bitwise LZ-78 compression algorithm with extension-order 2

pattern becomes 1111. This pattern has not been previously seen and thus, it will be added in the next available index, namely 5. The process continues until the entire input stream has been completely parsed.

When the input stream has totally parsed into subsequences, the algorithm represents each subsequence in the dictionary except the initialized ones as a pair (numerical value, n-bit). This representation is accomplished by splitting each subsequence into two parts. The first part is the subsequence itself except the least significant n-bit in it, namely sub-pattern. The second part is the least significant n-bit in the subsequence (which is called innovative phrase). The numerical value is the index of the sub-pattern in the subsequence.

In the example, shown in Fig. 2, the pair (numerical value, n-bit) of the subsequence of index 5, 1111, is demonstrated. The subsequence is separated into a sub-pattern, 11 and the innovative bits, 11. This sub-pattern is identical to the binary subsequence data exists at slot four in the dictionary. Therefore, we assigned the pair (4, 11) to the subsequence.

Now the BWLZ-78 algorithm determines the primary index. This index is the maximum numerical value in the pair representation of each subsequence in the dictionary. The primary index is crucial in determining the variable L. This variable is the maximum number of bits needed to represent the primary index. For instance, in the given example, the primary index is five and thus L is equal to 3.

Finally, each of the resulting pair is represented by a uniform block of L+n bits. The L bits are the binary representation of the numerical value (i.e., first coordinate) in the pair. The n-bit(s) are the innovation bit(s) available in the second coordinate in the same pair. For instance, in Fig. 2, the pair (4, 11) is represented as 10011. In turn, the compressed file is obtained by concatenating the binary representation of all pairs.

Bitwise de-compression process: The de-compression side of the algorithm accepts a condensed stream and restores the original input one. The de-compression process starts by extracting the information available in the header file. The information includes the characters available in the original source sorted in descending order, extension order (n) and the variable L (Fig. 1).

Based on the extraction of the header file information, the received stream is divided into equal blocks of fixed size L+n. Thereafter, each block is converted back into pairs (numerical value, n-bit). The first coordinate is the decimal equivalent that corresponds to the first L bits. This decimal value is the index of the root subsequence. The second coordinate is the least significant n-bit (i.e., innovation bit(s)).

The next step in the de-compression process is a dictionary construction. Firstly, the dictionary is pre-filled with the same initialization subsequences that are used in the compression process. As was previously mentioned, the extension-order (n) determines the number of initialization subsequences. The binary data in the next free row in the dictionary results from concatenating the root subsequence and the innovation bit(s). The de-compression process continues until the last original binary data in the compressed file retrieved back.

Figure 2 shows an example that illustrates the entire de-compression process. In the example, the compressed stream is divided into 5 bits blocks. Thereafter, each block is converted into a pair format. The corresponding pair format to the block 10011 is (4, 11). Eventually, the binary data in row 5, 1111, is obtained by concatenating the root subsequence at index 4, 11, with the innovation bits 11.

Finally, the equivalent binary file restores back. Now, to retrieve the original ATF, the binary file is simply divided into fixed-length codewords of eight bits each. In turn, each eight-bit codeword is remapped into its corresponding character in according with the header file information.

SECURITY OF ARABIC COMPRESSED FILE

One of the characteristics of the proposed technique is that it naturally places embedded security on the compressed file. The security is based on auto-generation of the mapping table, extension-order (n) and the length of the binary encoded representation (L). These parameters collectively are stored in a header file shared between the compression process and the de-compression one. The header file forms a private key. It is noteworthy that the frequency of occurrence for each Arabic character depends solely on the input file, i.e., a character may have different assigned codes depending on its frequency in the input file. For instance, the character "Meem" in Table 1 is assigned to the code 11111011. The same character in Table 2 is assigned to the code 11110111.

EXPERIMENTAL RESULTS AND DISCUSSION

To ensure the faithfulness of the proposed compression technique, two ATFs with different sizes has randomly selected. The performance of the proposed technique is compared with the traditional LZ algorithms. The comparison has been performed based on two mapping schemes, ASCII and dynamic mapping scheme.

The compression ratio as well as the compression/decompression speed at any extension order (n) is measured. Figure 3 and 4 show the performance

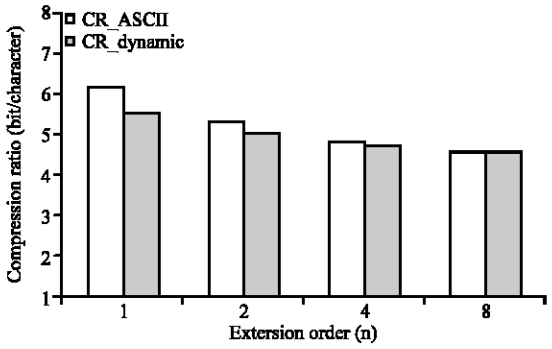

Fig. 3: Compression ratio (bits/character) versus the extension order (n) of الرحيق المختوم.txt file

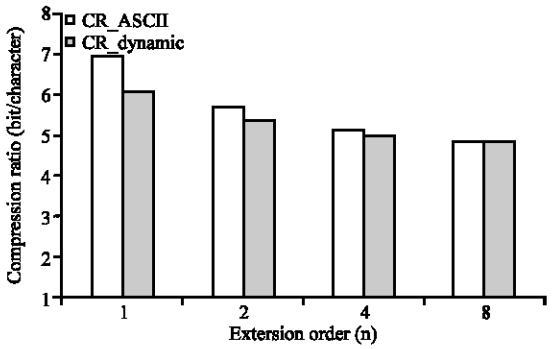

Fig. 4: Compression ratio (bits/character) versus the extension order (n) of تحفة العروس.txt file

comparison of traditional Lempel-Ziv and the proposed technique in terms of compression ratio for different extension-orders (e.g., n =1, 2, 4 and 8).

In Fig. 3, it is easy to see that the adaptive compression technique outperforms the other compression techniques that manipulate the generated binary file based on using the conventional ASCII code. Furthermore, one can observe that the BWLZ-78 algorithm of the fourth-order extended binary source (i.e., a source with 16 symbols) achieves a compression ratio close to that of the conventional LZ-78 algorithm.

To scrutinize in-depth the performance of applying BWLZ-78 algorithm on ATF, another file has been manipulated. This file is entitled تحفة العروس.txt with a size of 413 kB. The amount of compression displayed in Fig. 4 shows that the performance of the compression technique is consistent with that one of الرحيق المختوم.txt (size of 765 kB) file at any extension order (n).

CONCLUSIONS

The proposed technique can be utilized to reduce the volume of Arabic data to be exchanged through medium or stored on disk. The efficiency of the new technique stems from the fact that it is file-dependent. As a result, each file is condensed using its own character mapping table. It is noteworthy that the construction of such a table increases the overall time complexity of the compression process. However, this slight increment in time complexity will amortize by the efficiency that the proposed technique achieved. In addition, we do believe that the advancements in technology will reduce the impact of this complexity.

Moreover, the proposed technique increases the security of the compressed file. This is achieved by using the mapping table as a private key. In sum, the results motivated the researchers to apply the proposed algorithm to other data types such as voice and image.

REFERENCES

Bell, T.C., J.G. Cleary and I.H. Witten, 1990. Text Compression. Prentice Hall, Englewood Cliffs, Reading, New Jersey.

Elabdalla, A.R. and M.I. Irshid, 2001. An efficient bitwise Huffman coding technique based on source mapping. Comput. Electrical Eng., 27: 265-272.

Held, G. and T. Marshall, 1991. Data Compression. John Wiley and Sons, New York.

Jaradat, A.R. and M.I. Irshid, 2001. A simple binary run-length compression technique for non-binary sources based on source mapping. Active Passive Electronic Components, 24: 211-221.

Jaradat, A.R.M., M.I. Irshid and T.T. Nassar, 2006a. A file splitting technique for reducing the entropy of text files. Int. J. Inform. Technol., 3: 109-113.

Jaradat, A.R.M., M.I. Irshid and T.T. Nassar, 2006b. Entropy reduction of Arabic text files. Asia J. Inform. Technol., 5: 578-583.

Kanaan, G., R. Al-Shalabi and S. Ghwanmeh, 2006. Efficient data compression scheme using dynamic Huffman code applied on Arabic language. J. Comput. Sci., 2: 885-888.

Kida, T., M. Takeda, A. Shinohara and S. Arikawa, 1999. Shift-And Approach to Pattern Matching in LZW Compressed Text. In: Combinatorial Pattern Matching, Crochemore, M. and M. Paterson (Eds.). LNCS. 1645, Springer-Verlag, Berlin, Heidelberg, ISBN: 978-3-540-66278-5, pp: 1-13.

Musa, A. and M. Irshid, 2008. A modified text compression technique based on Lempel-Ziv algorithm. Proceedings of the International Conference on Information and Knowledge Engineering, (ICIKE'08), Las Vegas, Nevada, pp: 515-522.

Musa, A., A. Al-Dmour, O. Al-Khaleel and M. Irshid, 2008. An efficient text compression technique using Lempel-Ziv algorithm. Proceedings of the International Conference on Modeling and Simulation Conference, Nov. 18-20, Petra-Jordan, pp: 72-76.

Powell, M., 2009. Evaluating lossless compression methods. http://corpus.canterbury.ac.nz/research/evaluate.pdf.

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

Sayood, K., 2000. Introduction to Data Compression. 2nd Edn., Morgan Kaufmann, San Francisco, USA., ISBN: 1558605584, pp: 636.

Zeeh, C., 2009. The Lempel Ziv algorithm. http://tuxtina.de/files/seminar/LempelZivReport.pdf.