INTERNATIONAL JOURNAL OF ADVANCED STUDIES
IN COMPUTER SCIENCE AND ENGINEERING
IJASCSE VOLUME 5 ISSUE 11, 2016

11/30/2016

Enhancing the Compression Ratio of the HCDC Text Compression Algorithm

Hussein Al-Bahadili and Ghassan F. Issa
Faculty of Information Technology
University of Petra
Amman, Jordan
hbahadili@uop.edu.jo, gissa@uop.edu.jo

Ahmed Imad
Faculty of Information Technology
Middle East University
Amman, Jordan
ahmed.emad@gmail.com

***Abstract*— The Hamming Codes based Data Compression (HCDC) algorithm is a lossless data compression algorithm based on the concept of the error-correction Hamming Codes. The HCDC algorithm has been used successfully in a number of text compression applications providing a competitive compression ratio to existing well-known text compression algorithms. This paper describes an enhanced version of the HCDC algorithm, which is referred to as E-HCDC algorithm. The E-HCDC utilizes a new encoding scheme called the *m*-encoding scheme, where *m* represents the maximum length for the pre-fix bits. The compression ratios achieved by the E-HCDC are calculated for a number of text files of various sizes from widely-used corpora, and it reveals an improvement of 15% to 20% over the original HCDC algorithm.**

***Keywords- Text compression; Hamming Codes; HCDC algorithm; compression ratio*.**

I. INTRODUCTION

Data compression algorithms are developed to reduce the size of source data by substituting its original symbols with shorter symbols or by removing redundancy from the source data, so that it requires less disk space for storage, and less bandwidth and power consumption when transmitted over data communication channels [1]. Data compression consists of two main algorithms: compression and decompression algorithms. The first comprises algorithmic transformations of a source data representation to produce compact representation, while the second comprises algorithmic transformations to restore an exact or an approximate representation of the source data [2]. As a result of that data compression algorithms are classified into two different styles based on the conformity of the decompressed data, these are: lossless and lossy compression.

Lossless algorithms restore an exact copy of the source data (i.e., the source and the decompressed files are identical), so they are used for text files, executable codes, database files compression. Lossless algorithms usually achieve a compression ratio ranging between 2 to 8. Examples of lossless algorithms include: Huffman coding, Adaptive Huffman coding, Shannon-Fano Coding, Run Length Encoding, Arithmetic coding, and LZ77 and LZ78 families [1, 3].

Lossy algorithms cannot restore an exact copy of the source data and usually restore an approximate copy of the source data. This type of compression is used in applications where the source and the decompressed data are not necessary to be identical, such as multimedia streaming, video conferencing, VoIP applications, and MMS messaging. It achieves a data compression ratio up to 200, depending on the type of compressed data. A higher compression ratio can be achieved if more variations are allowed to be introduced in the decompressed data [1].

A novel lossless data compression algorithm based on the error correcting Hamming codes, namely, the Hamming Codes based Data Compression (HCDC) algorithm was developed by Al-Bahadili [4] and used successfully by a number of researchers for text and audio data compression applications [5-7]. In the HCDC algorithm, the source data is first converted to a binary sequence, which is divided into blocks of n-bit length. To utilize the Hamming codes, each block is considered as a Hamming codeword consisting of d data bits and p parity bits (where n=d+p, and $p=\lceil ln(d)/ln(2)\rceil+1$). According to Hamming codes, these codewords are categorized into 2^d valid codewords and 2^n-2^d non-valid codewords. For a valid codeword, only the d data bits preceded by "0" are written to the compressed file, while for a non-valid codeword all n bits preceded by "1" are written to the compressed file. These additional "0" and "1" bits are added to distinguish the valid and the non-valid blocks during the decompression process.

For Standard English text compression, the HCDC algorithm was implemented as follows: each character is treated as a 7-bit Hamming codeword having n=7, d=4, and p=3, the frequency of each character (f_i, i=1 to N_c, where N_c is the number of character sets in the text file) is calculated and the characters are sorted from the most common character (MCC) to the least common character (LCC). Then, the 16 MCCs are considered as valid codewords and replaced with 5 bits instead of the original uncompressed 7-bit ("0" followed by 4-bit representing the sequence number of the character 0 to 15), while the remaining characters are considered as a non-valid codewords replaced with b+1 bits

instead of the original uncompressed 7-bit ("1" followed by *b*-bit representing the sequence number of the character 0 to N_c-16, and $b=\lceil ln(N_c-16)/ln(2)\rceil$). It can be seen from the above discussion that the HCDC algorithm utilizes a simple encoding scheme for the pre-fix bit.

This paper develops a new encoding scheme for the pre-fix bits, which is used to develop an enhanced version of the HCDC algorithm with higher compression ratio, namely, the E-HCDC algorithm. The new prefix encoding scheme is called the *m*-encoding scheme, where *m* represents the length of the longest pre-fix bits. The compression ratio of the E-HCDC algorithm is calculated and compared with the compression ratio of the original HCDC algorithm for a number of text files from Standard English corpora.

This paper is divided into five sections. This section provides an introduction to the main theme of the paper, and also introduces the problems statement and the objectives of this research. The rest of this paper is organized as follows. Section II reviews some of the most recent and related work. The E-HCDC algorithm and its core component, the *m*-encoding scheme, are described in Section III. Section IV compares the compression ratios achieved by the HCDC and E-HCDC algorithms over a number of text files from Standard English Text Corpora. Finally, Section V presents the main conclusions of this research and also point-out future research direction.

II. Literature Review

A large number of data compression algorithms have been developed and used throughout the years. Some of which are of general use, i.e., can be used to compress files of different types (e.g., text files, image files, video files, etc.). Others are developed to compress efficiently particular types of files [1]. In this work, we concern with text compression, which has to be lossless so that exact form of the original data can be retrieved back.

Al-Bahadili [5] developed a lossless bit-level data compression algorithm based on the error-correcting Hamming codes, and consequently it was referred to as the HCDC algorithm. The theoretical analysis of the algorithm indicates that the algorithm reserved a tremendous potential and as a result of that it has been used by many researchers in many applications. Al-Bahadili and Rababa'a [5] investigated the performance of the HCDC algorithm for text compression application. They developed a scheme that utilizes the algorithm, which consists of six steps, some of which are applied repetitively to enhance the compression ratio of the algorithm, which is called the HCDC(*k*) scheme, where *k* refers to the number of repetition loops. The repetition loop continues until inflation is detected. The final compression ratio is a multiplication of the compression ratios of the individual loops. The results obtained for the HCDC(*k*) scheme demonstrated that the scheme has a higher compression ratio than most well-known text compression algorithms, and also exhibits a competitive performance with respect to many widely-used state-of-the-art compression tools.

Al-Bahadili and Al-Saab used the HCDC algorithm in developing a novel Web search engine model, namely, the Compressed Index-Query (CIQ) model [6]. The model incorporates two compression layers both based on the HCDC algorithm and implemented at the back-end processor side of the Web search engine. The test results demonstrated that the new CIQ model achieves 100% agreement with current uncompressed Web search engine models, a compression ratio of more than 1.3, and a speed up factor of more than 1.3 providing a reduction in processing time of more than 24%.

Amro et al. [7] used the HCDC algorithm for speech compression in Voice over Internet Protocol (VoIP) applications, in particular, they implemented the HCDC(*k*) algorithm with *k*=4 to achieve a significant compression ratio in the coefficient codebook without any scarification in the original Code-Excited Linear Prediction (CELP) signal quality since compression is lossless.

Al-Bahadili & Hussain [8] developed a bit-level data compression algorithm in which the binary sequence is divided into blocks of *n*-bit length, which gives each block possible decimal values between 0 to 2^n-1. The decimal values of all blocks are calculated and if the number of different decimal values (*d*) is ≤256, then the binary sequence can be compressed using *n*-bit character wordlength. Otherwise, the process continues until finding a value of *n* that gives *d*≤256. Thus, a compression ratio of approximately *n*/8 can be achieved. They referred to this algorithm as the Adaptive Character Wordlength (ACW) algorithm, since the compression ratio of the algorithm is a function of *n*, it is abbreviated as ACW(*n*) algorithm, which was used to develop a more efficient version called the ACW(*n*,*s*) algorithm, where *s* is the number of subsequences into which the original sequence is divided [9].

Khancome [10] proposed a full text compression algorithm, the source data is divided into suitable blocks; as well as, all characters in each block are assigned their positions. Afterwards, these positions are converted into the bit form. Empirically, the algorithm is efficient for the source data which takes several bytes per one character. The experimental results showed that the source data could be saved 11.50% in minimum; meanwhile, the maximum is 76.56%. Nicolae et al. [11] introduced a lossless non-reference-based compression algorithm named lossless FastQ compressor. They compared the algorithm with other state-of-the-art big data compression algorithms. The comparison reveals that the algorithm achieves better compression ratios.

Nofal [12] proposed a bit-level files compression algorithm providing a maximum of 10% compression ratio. Jaradat et al. [13] proposed a file splitting technique for the reduction of the n^{th}-order entropy of text files. Barr and Asanovic [14] studied the energy savings possible by lossless compressing data prior to transmission. Because

wireless transmission of a single bit can require over 1000 times more energy than a single 32-bit computation. The overall energy to send and receive data can be reduced by 11% compared with a well-chosen symmetric pair, or up to 57% over the default symmetric scheme.

Irshid [15] proposed a very simple and efficient binary run-length compression algorithm, which is based on mapping the non-binary information source into an equivalent binary source using a new fixed-length code instead of the ASCII code. The codes are chosen such that the probability of one of the two binary symbols; say 0, at the output of the mapper is made as small as possible. Moreover, the "all 1's" code is excluded from the code assignments table to ensure the presence of at least one 0 in each of the output codewords. Compression is achieved by encoding the number of 1's between two consecutive 0's using either a fixed-length or a variable-length code. When applying this simple encoding technique to English text files, they achieve a compression of 5.44 bpc and 4.6 bpc for the fixed-length code and the variable length (Huffman) code, respectively.

III. The Enhanced HCDC Algorithm

The main objective of this work is to increase the compression ratio of the HCDC algorithm [4]. To do so, a new pre-fix encoding scheme is developed, namely, the *m*-encoding scheme. In this scheme, different pre-fix bit combinations are used depending on the types of characters presented in the text file and frequencies of these characters. This is in replacement of the "0"/"1" pre-fix encoding scheme of the HCDC algorithm. The enhanced version of the HCDC algorithm is referred to as the Enhanced HCDC (E-HCDC) algorithm, which is expected to achieve higher compression than the original HCDC algorithm.

A. The m-Encoding Scheme

In the *m*-encoding scheme, the characters frequencies are found first, and then the characters are sorted according to their frequencies from the MCC to the LCC. The characters are divided into groups of 16 characters each starting from the MCC. The last group may contain 16 or less characters depending on the number of characters sets in the source text. Since, the maximum number of characters for 7-bit character length is 128 characters; then the maximum number of groups is 8 numbered from G_1 to G_8. The characters for each group are numbered from 0 to 15 and assigned a binary code from 0000 to 1111.

For each group, we will assign different combination of pre-fix bits; e.g., the pre-fix for G_1 is "0" similar to the original HCDC. The pre-fix bit for the other groups are 10, 110, 1110, and so on until the last group which will have all 1's. Thus, we will have 7 1's for G_8. For each group we will add 4-bit representing the sequence number of the character within the group. The pre-fix bits and the numbers of bits for each group are summarized in Table I.

It can be seen from Table I that we are reducing the number of bits written to the compressed data file to 6-bit and 7-bit for G_2 and G_3 in comparison to 8-bit for the HCDC algorithm. While it is kept unchanged with 8-bit for G_4, and start increasing by 1-bit for each of the remaining groups until it reaches 11 for G_8.

The above discussion shows that more bits are required to represent the characters in the last four groups (G_5 to G_8), but, fortunately, the characters frequencies within these groups are small in comparison to that in the top 4 groups (G_1 to G_4). Furthermore, most of the time we have the characters span over not more than 6 groups, which means that the maximum number of bits required to represent the compressed characters is 9 bits, because the number of pre-fix bits for the last group is equal the number of bits for previous one but with pre-fix bits set to 1's.

This form of pre-fix encoding guarantees higher compression ratio than for the HCDC algorithm even for files with flat characters frequencies distribution. However, in this work, to improve the performance further, we introduce a modification to the above continuous encoding.

TABLE I THE PRE-FIX BITS AND THE NUMBER OF BITS WRITTEN TO THE COMPRESSED FILE

Group (*G*)	Pre-fix bits	No. of Pre-fix bits	No. of Compressed Bits	
			E-HCDC	HCDC
1	0	1	5	5
2	10	2	6	8
3	110	3	7	8
4	1110	4	8	8
5	11110	5	9	8
6	111110	6	10	8
7	1111110	7	11	8
8	1111111	7	11	8

In the modified encoding, the number of pre-fix bits is limited to *m*-bit. Therefore, we call this pre-fix encoding as *m*-encoding. Before, we proceed with the description of this new encoding; let us define some parameters. The first one is the number of groups (G), which is calculated as:

$$G = N_c / 2^d \quad (1)$$

Where G is the number of groups, N_c is the number of characters sets, and d is the number of data bits in Hamming codeword. Second, the number of characters within the last group L_g, which can be calculated as:

$$L_g = N_c - 2^d \times (G\text{-}1) \quad (2)$$

Third, the number of bit (b) required to represent the characters within the last group, which can be calculated as:

$$b = \left\lceil \frac{\ln(L_g)}{\ln(2)} \right\rceil \quad (3)$$

When m is selected to be 4-bit, then the pre-fix combinations are 0 for G_1, 10 for G_2, 110 for G_3, 1110 for G_4, and 1111 for the remaining group(s). So that the number of bits for the last groups can also be calculated as:

$$L_g = N_c - m \times 2^d \quad (4)$$

Table II shows the pre-fix bits combinations, the characters codes, and the compressed bits written to the compressed file for a file contains up to 96 characters. It is clear from the above discussion that the characters in G_5 and G_6 are merged together forming a group of 32 characters requiring 5-bit to represent each character from character 1 (00000) to character 31 (11111). The pre-fix bits are 4-bit (1111), and then each character will be written as 9-bit character to the compressed file. However, if $N_c \leq 80$ and $N_c > 64$, then $1 < L_t \leq 16$, and in this case b_t is ≤ 4, which means the number of compressed bits written to the compressed file is ≤ 8

TABLE II. THE PRE-FIX BITS AND THE NUMBER OF BITS WRITTEN TO THE COMPRESSED DATA FILE.

Group (*G*)	Character (*E*)	Pre-fix bits	No. of Pre-fix bits	Character Code	Compressed Bits	No. of Compressed Bits	
						E-HCDC	HCDC
1	1	**0**	**1**	0000	**0**0000	**5**	**5**
	:	**0**		:	:		
	:	**0**		:	:		
	16	**0**		1111	**0**1111		
2	17	**10**	**2**	0000	**10**0000	**6**	**8**
	:	**10**		:	:		
	:	**10**		:	:		
	32	**10**		1111	**10**1111		
3	33	**110**	**3**	0000	**110**0000	**7**	**8**
	:	**110**		:	:		
	:	**110**		:	:		
	48	**110**		1111	**110**1111		
4	49	**1110**	**4**	0000	**1110**0000	**8**	**8**
	:	**1110**		:	:		
	:	**1110**		:	:		
	64	**1110**		1111	**1110**1111		
5	65	**1111**	**4**	00000	**1111**00000	**9**	**8**
		1111					
	:	**1111**		:	:		
	:	**1111**		:	:		
6	:	**1111**		:	:		
	:	**1111**		:	:		
		1111					
	96	**1111**		11111	**1111**11111		

One important feature of the m-encoding is that for m=1, it behaves exactly the same as the HCDC, where in this case the first 16 MCCs (G_1) are preceded by 0, while all other characters (groups) are preceded by 1. However, there is only one difference, which is in the HCDC, all 7-bit character is appended to the compressed file, while here, only b bits are appended, and b is calculated depending on the sequence number of the character excluding the first 16 MCC. In this case, we may need less than 7-bit to represent characters in the compressed file. This provides further saving and increasing the compression ratio.

One other important feature of the m-encoding scheme is the optimum value of m can be calculated automatically depending on the characters frequencies or characters counts. Using the m-encoding scheme, the size of the compressed binary sequence can be calculated as:

$$Q_b = (d+1)\sum_{i=1}^{2^d} T_i + \sum_{j=2}^{m}(d+j)\sum_{i=1+(j-1)\times 2^d}^{j\times 2^d} T_i + (b_t+m)\sum_{i=1+m\times 2^d}^{N_c} T_i \quad (5)$$

Where Q_b is length of the compressed binary sequence in bits, d is number of data bits in the Hamming codeword, T_i is counts for character i (i=1 to N_c), m is maximum length for the pre-fix bits, b_t is number of bits representing the characters of the last group, and N_c is number of characters sets in the text file. The code can calculate the values of Q_b for all possible values of m, and select the value of m that gives minimum Q_b to ensure highest possible compression ratio.

B. The E-HCDC Algorithm

This section presents the detailed description of the E-HCDC compression and decompression procedures for text compression.

The E-HCDC compressor

The E-HCDC compressor consists of the following steps:

1) Read-in the input uncompressed text file.
2) Find the characters sets in the uncompressed file (N_c), characters counts (T_i), and characters frequencies (f_i), where i=1 to N_c.
3) Sort the characters set from the MCC to the LCC.
4) Determine the optimum value for m.
5) Initialize the compressed binary sequence to Null.
6) Start the compression process by reading in one character at a time until processing all characters; find the character sequence number, the group it's belonging to, and its sequence within the group. According to the value of m and the group its belonging to, append the pre-fix bits to the compressed binary sequence, find the character's equivalent binary representation based on the associated character wordlength (d or b depending on m and G), and then append the character binary representation to the compressed binary sequence.
7) In order to be able to convert the resultant compressed binary sequence (Q_b) to 8-bit character length, and because Q_b may not be a multiple of 8, then we append padding bits (say a 0's) at the end of the compressed binary sequence, and the length of the padding bits (a) need to be stored at the compressed file header, so these bits can be discarded during the decompression phase.
8) Construct the compressed file header and append it to the compressed text sequence, which should contain all information necessary during the decompression process.
9) Convert the compressed binary sequence to characters by reading-in 8-bit at a time until converting all compressed binary sequence. For each 8-bit, find its equivalent ASCII code, and then append the equivalent character to the compressed text sequence.
10) Write the compressed text sequence to the compressed file.

The E-HCDC decompressor

The E-HCDC decompressor consists of the following steps:

1) Read-in the input compressed file.
2) Extract the file header, and find-out N_c, the uncompressed characters themselves (U_i) (where i=1 to N_c.), the maximum length of the pre-fix bits (m), and the number of padding bits (a).
3) Determine the number of groups (G) as G=m+1, the number of characters in the last group (L_t), and the number of bits to represent the characters of the last group (b).
4) Initialize the compressed binary sequence (Q_b) to Null.
5) Initialize the decompressed text sequence (D_t) to Null.
6) Extract the compressed text sequence and read-in it one character at a time, convert each character to a binary representation according to its ASCII code, and append the resultant binary representation to the compressed binary sequence (Q_b), until all characters are read-in.
7) Discard the appended bits (a).
8) Start the decompression process by reading in 1-bit, if this bit is 0, then the character belongs to G_1 (i.e., g=1), read-in the next 4-bit, find-out the equivalent decimal value (e) between 0 to 15, which represents the sequence number of the character (q) within G_1 minus 1 (q=e+1), determine the global sequence number (s) as:

$$s = q + 2^d (g - 1) = 1 + e + 2^d (g - 1) \quad (6)$$

Then, retrieve the character with sequence number s (U_s) from the uncompressed characters lists and append it D_t. If the read-in bit is 1 (count the number of 1's (z)), then reads-in another bit until a 0 is read-in or read-in a maximum of m 1's. In this case, the group number (g) is equal to z+1 as long as z+1<m, otherwise it is equal to m. If g is not the last group (g<G), then read-in 4-bit, otherwise (g is the last group g=G) read-in b-bit, and perform the same steps above and append another character to D_t. This process continues until reading all compressed binary sequence.

9) Write the decompressed text sequence (D_t) to the uncompressed file.

It can be clearly seen from the above discussion that the E-HCDC algorithm is:

1) *Bit-level*. It processes the data at a binary level.
2) *Lossless*. An exact form of the source file is retrieved.
3) *Adaptive*. The character-binary coding depends on the characters frequencies.
4) *Asymmetric*. The compression time is higher than the decompression time.

IV. Results and Discussions

The E-HCDC algorithm is implemented using VB.net programming language; however, it is important to indicate that the code is not optimized to determine the compression/decompression processing time, therefore, this paper only compares and shows the results for the compression ratio. It can be generally seen for the compression and the decompression procedures that the

algorithm can be classified as an asymmetric algorithm as the compression processing time looks longer than the time required for decompression.

This section compares the compression ratio of the E-HCDC ($C_{E\text{-}HCDC}$) against the compression ratio of the HCDC (C_{HCDC}) for a number of text files from standard corpora, namely, Calgary Corpus, Canterbury Corpus, Artificial Corpus, and Large Corpus [16]. Furthermore, a new parameter is introduced, which is referred to as the compression enhancement ratio (E_r) and it can be calculated as:

$$E_r = \frac{C_{E-HCDC} - C_{HCDC}}{C_{HCDC}} \times 100 \quad (8)$$

The results for C_{HCDC} and $C_{E\text{-}HCDC}$ are listed in Table III, which clearly shows that the E-HCDC algorithm enhances the performance of the HCDC algorithm by 15% to 20% for various files from different corpora. This is achieved due to the implementation of the new *m*-encoding technique. Furthermore, Table III shows the values of *G* and *m* for each compressed file.

TABLE III. COMPARISON OF C_{HCDC} AND $C_{E\text{-}HCDC}$.

Corpus	File Name	File Size (Byte)	N_c	C_{HCDC}	$C_{E\text{-}HCDC}$	G (*m*) *m*=*G*-1	E_r (%)
Calgary corpus	bib	111261	81	1.177	1.455	6 (5)	18
	book1	768771	82	1.269	1.537	6 (5)	19
	book2	610856	96	1.247	1.509	6 (5)	18
	paper1	53161	95	1.211	1.484	6 (5)	18
	paper2	82199	91	1.265	1.531	6 (5)	19
	paper3	46526	84	1.261	1.525	6 (5)	19
	paper4	13286	80	1.257	1.52	5 (4)	19
	paper5	11954	91	1.218	1.486	6 (5)	18
	paper6	38105	93	1.203	1.477	6 (5)	18
Canterbury corpus	alice29.txt	152089	74	1.258	1.529	5 (4)	19
	asyoulik.txt	125179	68	1.230	1.508	5 (4)	18
	lcet10.txt	426754	84	1.252	1.520	6 (5)	19
	plrabn12.txt	481861	81	1.269	1.537	6 (5)	19
Artificial corpus	alphabet.txt	100000	26	1.138	1.486	2 (1)	18
	random.txt	100000	64	0.969	1.236	4 (3)	15
Large corpus	bible.txt	4047390	63	1.294	1.551	4 (3)	19
	world192.txt	2473400	94	1.206	1.471	6 (5)	18

V. CONCLUSIONS

The main conclusions of this paper are: (1) the E-HCDC algorithm enhances the compression ratio of HCDC algorithm by 15% to 20%. This is achieved due to the utilization of the newly developed *m*-encoding scheme for the pre-fix bits, (2) the analytical procedure for estimating the compression ratio provides a mean to automatically compute the optimum length for the pre-fix bits, and (3) the E-HCDC algorithm achieves a compression ratio that is high enough to be competent with the compression ratio achieved by many well-known algorithms of statistical and adaptive nature.

The main recommendations for future work may include: (1) utilize the *m*-encoding to develop a repetitive version of the E-HCDC algorithm (E-HCDC(*k*)), (2) use the E-HCDC algorithm as a post processing technique to increase the compression ratio of statistical lossless data compression algorithms, such as Shanon-Fano coding, Huffman coding, arithmetic coding, a combination of these algorithms, or any modified form of them, and (3) develop an optimized version of the code to compare its runtime with other compression algorithms and state-of-the-art software, and compare the compression/decompression processing runtimes.

REFERENCES

[1] K. Sayood (2012). Introduction to data compression. Fourth Edition, Morgan Kaufmann.

[2] A. B. Sharma, L. Golubchik, R. Govindan, & M. J. Neely (2009). Dynamic data compression in multi-hop wireless networks. In Proceedings of the 11th International Joint Conference on Measurement and Modeling of Computer Systems, pp. 145–156.

[3] L. G. Rueda & B. J. Oommen (2006). A fast and efficient nearly-optimal adaptive Fano coding scheme. Journal of Information Sciences, Vol. 176, Issue 12, pp. 1656–1683.

[4] H. Al-Bahadili (2008). A novel lossless data compression scheme based on the error correcting Hamming codes. Computers & Mathematics with Applications, Vol. 56, Issue 1, 143–150.

[5] H. Al-Bahadili & A. Rababa'a (2010). A bit-level text compression scheme based on the HCDC algorithm. International Journal of Computers and Applications (IJCA), Vol. 32, Issue 3, pp. 48-52.

[6] H. Al-Bahadili & S. Al-Saab (2011). Development of a novel compressed index-query Web search engine model. International Journal of Information Technology and Web Engineering (IJITWE), Vol. 6, No. 3, pp. 39-56.

[7] I. Amro, R. Abu Zitar, & H. Al-Bahadili (2011). Speech compression exploiting linear prediction coefficients codebook and hamming

correction code algorithm. International Journal of Speech Technology, Vol. 14, No. 2, pp. 65-76.

[8] H. Al-Bahadili & S. M. Hussain (2008). An adaptive character wordlength algorithm for data compression. Computers & Mathematics with Applications, Vol. 55, Issue 6, pp. 1250–1256.

[9] H. Al-Bahadili & S. M. Hussain (2010). A bit-level text compression scheme based on ACW algorithm. The International Journal of Automation and Computing (IJAC), Vol. 7, No. 1, pp. 128-136.

[10] C. Khancome (2011). Text compression algorithm using bits for character representation. International Journal of Advanced Computer Science, Vol. 1, No. 6, pp. 215-219.

[11] M. Nicolae, S. Pathak, & S. Rajasekaran (2015). LFQC: A lossless compression algorithm for FASTQ files. Journal of Bioinformatics, Vol. 31, No. 20, pp. 3276-3281.

[12] S. Nofal (2007). Bit-level text compression. Proceedings of the 1st International Conference on Digital Communications and Computer Applications, pp. 486-488.

[13] A. R. M. Jaradat, M. I. Irshid, & T. T. Nassar (2007). A file splitting technique for reducing the entropy of text files. International Journal of Computer, Information, Systems and Control Engineering, Vol. 1, No 7, pp. 2097-2101.

[14] K. C. Barr & K. Asanovic (2006). Energy-aware lossless data compression. ACM Transaction on Computer System, Vol. 24, No. 3, pp. 250–291.

[15] M. I. Irshid (2001). A simple binary run-length compression technique for non-binary sources based on source mapping. Active and Passive Electronic Components, Vol. 24, No. 4, pp. 211-221.

[16] A. Ali (2013). Enhancing the compression ratio of the HCDC-based text compression scheme. M.Sc Thesis. Middle East University, Faculty of Information Technology, Amman-Jordan.