International Journal of Automation and Computing 7(1), February 2010, 123-131
DOI: 10.1007/s11633-010-0123-6

A Bit-level Text Compression Scheme Based on the ACW Algorithm

Hussein Al-Bahadili[1] Shakir M. Hussain[2]
[1]Arab Academy for Banking and Financial Sciences, Amman 11942, Jordan
[2]Faculty of Information Technology, Petra University, Amman 11196, Jordan

Abstract: This paper presents a description and performance evaluation of a new bit-level, lossless, adaptive, and asymmetric data compression scheme that is based on the adaptive character wordlength (ACW(n)) algorithm. The proposed scheme enhances the compression ratio of the ACW(n) algorithm by dividing the binary sequence into a number of subsequences (s), each of them satisfying the condition that the number of decimal values (d) of the n-bit length characters is equal to or less than 256. Therefore, the new scheme is referred to as ACW(n, s), where n is the adaptive character wordlength and s is the number of subsequences. The new scheme was used to compress a number of text files from standard corpora. The obtained results demonstrate that the ACW(n, s) scheme achieves higher compression ratio than many widely used compression algorithms and it achieves a competitive performance compared to state-of-the-art compression tools.

Keywords: Data compression, bit-level text compression, ACW(n) algorithm, Huffman coding, adaptive coding.

1 Introduction

Text compression is usually obtained by substituting a shorter symbol for an original symbol in the source data, which should contain the same information but with a smaller representation in length. Text compression often derives its benefit from the views of its textual content, where the content can be seen as a stream of syllables or words[1,2].

Transforming text characters into a stream of syllables or words is not an easy process. The transformation heavily influences almost all inner data structures, because they must be able to work with undefined (and often high) number of syllables or words instead of the original alphabet of 256 characters, which need to be exported from the encoder to the decoder[3−5].

Text compression can also be handled at bit-level, as each character has its specific binary representation. Little work has been done to exploit bit-level compression, and most of the existing work demonstrates limited advantages over other data compression algorithms of all types[6].

This paper presents and evaluates the performance of a new bit-level, lossless, adaptive, and asymmetric data compression scheme that is based on the adaptive character wordlength (ACW(n)) algorithm[7]. The new scheme introduces a number of modifications to the ACW(n) algorithm to enhance its performance. The main idea behind this scheme is to subdivide the binary sequence into a number of subsequences (s), each of them satisfies the condition of having the number of the different decimal values ($d \leqslant 256$). Therefore, we refer to this new scheme as ACW(n,s) scheme.

To further enhance the performance of the ACW(n,s) scheme, an efficient character-to-binary coding format, namely, the adaptive coding format, is introduced. This coding format yields a low entropy binary sequence so that it grants higher compression ratio.

The rest of this paper is organized as follows. Section 2 reviews some of the most recent bit-level data compression algorithms. In Section 3, a detailed description of the ACW(n) algorithm is given. The proposed ACW(n, s) scheme is described in Section 4. Experimental results that quantify the compression ratio of the new scheme, over a number of text files from standard corpora, are presented and discussed in Section 5. The results obtained are compared with a number of widely used data compression algorithms and state-of-the-art software. Finally, in Section 6, conclusions are drawn and recommendations for future work are pointed-out.

2 Literature reviews

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.)[2,4,8,9]. Others are developed to efficiently compress a particular type of files. This paper is concerned with text compression algorithms, which can be classified, according to the representation form of the data at which the compression process is performed, into two classes. These are: syllable or word based text compression algorithms and bit-level text compression algorithms.

Syllable or word-based text compression algorithms are beyond the scope of this paper; therefore; in this section, we shall only review some of the most recently developed bit-level text compression algorithms. On the other hand, detail descriptions and extensive literature reviews on syllable or word-based text compression algorithms can be found in [1–5, 10–13].

In [14], a novel lossless binary data compression algorithm based on the error correcting Hamming codes, namely, the Hamming codes based data compression

Manuscript received September 10, 2008; revised April 20, 2009

(HCDC) algorithm, was proposed. In this algorithm, the binary sequence is subdivided into blocks of n bits length, and each block is considered as Hamming codeword that consists of p parity bits and d data bits ($n = d + p$). The analytical analysis shows that a maximum compression ratio of n over d+1 can be achieved by this algorithm.

In [6], an adaptive bit-level text compression scheme that is based on the HCDC algorithm was introduced. The scheme consists of six steps, some of which are repetitively applied to achieve higher compression ratio. The repetition loops continue until inflation is detected and the compression ratio is equivalent to the multiplication of the compression ratios of the individual loops; therefore, the new scheme is referred to as HCDC(k), where k refers to the number of repetition loops. The maximum number of repetition loops before inflation detected was found to be $\leqslant 6$ loops and reached a 1.6–2.6 compression ratio.

A bit-level file compression algorithm was proposed in [15], in which a set of groups of bits were considered as minterms representing a Boolean function. Then, some algebraic simplifications were made on these functions to reduce the number of minterms and hence the size of the binary sequence. The maximum compression ratio achieved was not more than 10%.

An approach to universal noiseless compression scheme, based on error correcting codes, was presented in [16]. The scheme was based on concatenation of Burrows-Wheeler transform (BWT) with low-density parity-check (LDPC) code. The scheme has linear encoding and decoding times.

A fixed-length Hamming (FLH) algorithm was introduced in [17] as an enhancement to Huffman coding (HU) to compress text and multimedia files. The investigation and testing on these algorithms, on different texts and multimedia files, indicated that the FHL following HU and HU following FLH enhance the compression ratio by a factor of not more than 20 %.

A fast text compression with neural network model was introduced in [18]. It produces better compression than popular Limpel-Ziv compressors (zip, gzip, and compress), and is competitive in time, space, and compression ratio with prediction by partial matching (PPM) and BWT algorithms. It was concluded that it is practical to use neural networks for text compression in any application that requires a high speed.

3 The ACW(n) algorithm

In the ACW(n) algorithm[7], first, the source file is converted into binary sequence using a certain character- to-binary coding format (e.g., the well-known ASCII coding). The binary sequence is then subdivided into blocks of n-bit length. The equivalent decimal value for each block is calculated, and if the total different decimal values $d \leqslant 256$, compression can be performed using n-bit character wordlength. Otherwise, another value of n should be examined. This iterative process continues until a specific value of n satisfies the above condition; otherwise, the binary sequence cannot be compressed. Therefore, this algorithm is referred to as ACW(n) algorithm.

If a specific value of n satisfies the above condition, the different decimal values are sorted in an ascending or descending order such that each block is then converted into character according to its sequence number but not according to its actual decimal value. In order not to make the binary sequence mixed up during the decompression process, these sorted values should be stored with other information (e.g., n, d) at the compressed file header.

The probability of satisfying the condition of $d \leqslant 256$ is inversely proportional to n and it is given as $p = 2^{8-n}$. For example, the probabilities of $d \leqslant 256$, for $n = 9$ and $n = 10$, are 0.5 and 0.25, respectively.

There main issues of the ACW(n) algorithm can be summarized as follows:

1) The binary sequence can be compressed using $n-$bit character wordlength only if $d \leqslant 256$.

2) The probability to successfully compress the binary sequence using n-bit character wordlength is inversely proportional to n.

3) Finding the optimum value for n that provides the maximum compression ratio is a time-consuming process, especially for large binary sequences.

4) Converting text to binary using the characters ASCII code yields a high entropy binary sequence, which means that either little or no compression can be achieved.

These issues can be considered as drawbacks that may degrade the performance of the ACW(n) algorithm.

4 The ACW(n, s) scheme

In order to enhance the performance of the ACW(n) algorithm and tackle all consequences of the above issues, the binary sequence is subdivided into a number of subsequences (s), each of them satisfying the condition that $d \leqslant 256$. Therefore, the new scheme is referred to as ACW(n, s). By subdividing the original binary sequence into subsequences, we can eliminate the first downside of $d \leqslant 256$ and consequently the worry of having low success probability, if large value of n is chosen.

To further enhance the performance of the ACW(n, s) algorithm, an adaptive coding format was introduced in which the original characters were coded to binary according to their probability of occurrence. This coding format reduces the entropy of the binary sequence so that a higher compression ratio is granted. Definition of entropy and the way the adaptive coding is working are given below.

In information theory, entropy is a measure of the information content of a message, where a message is defined as a stream of characters or symbols. The entropy of a character (symbol) is represented as the negative logarithm of its probability or frequency of occurrence within the message. Consequently, the entropy of a complete message would be the sum of the entropies of the individual characters, which is expressed as[19]

$$H = -\sum_{i=1}^{d} p_i \log_2 p_i \qquad (1)$$

where H is the entropy in bits, and p_i is the estimated probability of occurrence of each character within the message.

In adaptive coding, first, the character frequencies are calculated and sorted in a descending order from the most common character (MCC) to the least common character

(LCC). Second, the MCC is assigned a sequence number (SN) of 0, while the LCC is assigned an SN of $d-1$. Then, each character is converted into binary according to the binary representation of its SN. For example, consider the coding of the message "ababac". The message contains three characters ($d = 3$) and their frequencies of occurrence are as follows: three "a", two "b", and one "c". So, the MCC is "a" and the LCC is "c". Consequently, "a" is assigned an SN of 0, while "b" and "c" are assigned SNs of 1 and 2, respectively. Therefore, using adaptive coding, the equivalent binary code of "a" would be the 7-bit binary representation of the number 0, which is 0000000. Similarly, the equivalent binary codes of "b" and "c" are 0000001 and 0000010, respectively. Table 1 shows the frequency of occurrence of each character and its ASCII and adaptive equivalent codes.

Table 1 Character codes of the message "ababac"

Character	No. of characters	ASCII code	Adaptive code
a	3	1100001	0000000
b	2	1100010	0000001
c	1	1100011	0000010

Table 2 presents the message equivalent binary code using ASCII and adaptive codes, the length (N) of the binary sequence in bits, and the number of "0" and "1" in each sequence. Thus, applying (1), with $c = 2$ (we have only two symbols "0" and "1"), and $p(0)/p(1)$ is the number of "0"/"1" divided by N, yields entropies of 68.86 % and 25.73 % for the ASCII and the adaptive coding formats, respectively.

Table 2 Binary sequences of the message "ababac"

Coding format	Message binary code	Message length N (bit)	Number of "0"	Number of "1"
ASCII	1000011 0100011 1000011 0100011 1000011 1100011	42	23	19
Adaptive	0000000 1000000 0000000 1000000 0000000 0100000	42	39	3

The ACW(n, s) scheme compression/decompression algorithms as shown as Algorithms 1.

Algorithm 1. The ACW(n, s) compressor

```
Construct the binary sequence using any character-to-binary coding format, e.g., ASCII coding, Huffman coding, adaptive coding, etc.
Do
  Select a character wordlength ($n$)
  Subdivide the binary sequence into blocks ($B$) of $n$-bit length and add padding bits ($g$) if the length of the last block is less than $n$-bit
  Set $s = 0$; //Subsequences counter
  Set $m = 0$; //Total blocks counter
  Do
    Set $s = s + 1$;
    Set $d = 0$;//Different decimal values counter
    Set $k = 0$;//Blocks counter for each subsequence
    Do
      Read $n$-bit block
      $m = m + 1$;
      $k = k + 1$;
      Compute the equivalent decimal value ($D$)
      Search for $D$ in the computed $d$-1 values
      If ($D$) have not found, then
        Set $d = d + 1$;
        ...
    Loop until ($d > 256$) or ($m \geq B$) //$B$ is the total number of blocks
    Process these different decimal values and construct the subsequence header;
    ...
  Loop until ($m \geqslant B$)
  Compute the compression ratio.
Loop until ($n > n_{\max}$) //$n_{\max}$ is the maximum value for $n$ set by the user
Find the optimum character wordlength ($n$)
Construct the compressed file header
Re-read the binary sequence in blocks of $n$-bit character wordlength
Convert each block into characters using its sequence number and store them into the compressed file.
```

Algorithm 2. Algorithm for the ACW(n,s) decompressor

```
Read the compressed file header to define the values of $V$, $F$, $n$, $g$, $s$, $L[\cdot]$, $N[\cdot]$
For ($i$ =; $i \leqslant s$; $i$++) //Loop over all subsequences
{
  For ($j$ =; $j \leqslant L[i]$; $j$++) //$L[i]$ is the number of blocks within the $i$-th subsequence
  {
    Read a data character
    Find its equivalent decimal number
    Use this decimal value as a sequence number to retrieve the equivalent decimal value stored in the subsequence header
    Convert this decimal value into n-bit block
  }
}
Convert the binary sequence into characters according to the coding format and store them in the decompressed file.
```

4.1 The compressed file header

In order not to make the codes mixed up during the decompressing process, some information needs to be stored in the compressed file header as shown in Fig. 1. The size of the header is directly proportional to the number of subsequences; therefore, an optimization mechanism is required in finding the values of n and s, which yields a maximum compression ratio.

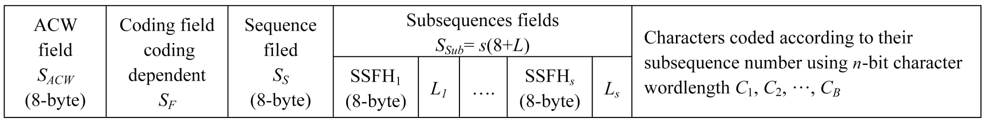
ACW field S_{ACW} (8-byte)
Coding field coding dependent S_F
Sequence filed S_S (8-byte)
Subsequences fields $S_{Sub}= s(8+L)$
SSFH$_1$ (8-byte)
L_1
....
SSFH$_s$ (8-byte)
L_s
Characters coded according to their subsequence number using n-bit character wordlength C_1, C_2, $\cdots$, C_B

Fig. 1 The compressed file using the ACW(n,s) scheme

The compressed file header of the ACW(n, s) scheme consists of the following fields:

1) ACW field (8-byte),

2) Coding field (coding format dependent),

3) Sequence field (8-byte),

4) Subsequence fields (ACW(n,s) dependent).

4.1.1 ACW field (S_{ACW})

The ACW field (S_{ACW}) is an 8-byte field enclosing information related to the ACW(n,s) algorithm, such as algorithm name (ACW), algorithm version, coding format, and character wordlength. The constituents of this field are kept similar to the first 8-bytes in the ACW(n) algorithm. Table 3 lists the constituents of this field and their description.

Table 3 Compressed file header

Field	Length	Description
ACW	3	Name of algorithm
V	1	Version of the algorithm
F	1	Coding format
Char(n)	1	The character wordlength
Char(d)	1	The number of the different decimal values
Char(g)	1	The number of padding bits added to the last block
D_1 to D_d	L	The actual decimal values the blocks may have

4.1.2 Coding field (S_F)

The coding field (S_F) encloses information related to the coding format, so that their content depends on the coding format indicated in the ACW field. This field is not required for the ASCII coding format. For the adaptive coding, it contains the number of characters within the source file and a list of these characters after sorting (N_c+1), while for Huffman coding, it enfolds the same components as in standard Huffman compression algorithm.

4.1.3 Sequence field (S_S)

The sequence field (S_S) is an 8-byte field that contains the number of subsequences in the first 4-byte, and the last 4-byte remain unused for future development.

4.1.4 Subsequence fields (S_{Sub})

The subsequence fields (S_{Sub}) enfold the subsequences information. It consists of two subfields. The first one is a fixed 8-byte subfield that includes the subsequence number (4-byte) and the number of the different decimal values within the subsequence (d) (1-byte). Three bytes are kept unused for future development.

The second subfield has a variable length, which is referred to as L, and it contains different decimals values of the blocks. The total length of the subsequence fields is given by

$$S_{Sub} = s(8 + L) \tag{2}$$

$$L = \begin{cases} \dfrac{2^n}{8}, & \text{for } n \leqslant 11 \\ d\left\lceil \dfrac{n}{8} \right\rceil, & \text{for } n \geqslant 12 \end{cases} \tag{3}$$

As shown in Fig. 1, the size of the compressed file header (S_H) in bytes is given by

$$S_H = S_{ACW} + S_F + S_S + s(8 + L). \tag{4}$$

Since both S_{ACW} and S_S are fixed 8-byte fields, (4) can be simplified into

$$S_H = 16 + S_F + s(8 + L). \tag{5}$$

Thus, as shown in Fig. 1, the size of the compressed file (S_c) is given by

$$S_C = S_H + B \tag{6}$$

where B is a positive integer number representing the number of blocks into which the binary sequence is subdivided. For a binary sequence of N-bit length, B is calculated by

$$B = \left\lceil \frac{N}{n} \right\rceil. \tag{7}$$

If the length of the last block is less than n-bit, then it should be padded with "0". The number of padding bits (g), which is added to the last block, is calculated by

$$g = B \cdot n - N. \tag{8}$$

Substituting (5) and (7) for S_H and B, respectively, into gives

$$S_c = 16 + S_F + s(8 + L) + B. \tag{9}$$

4.2 Computing the compression ratio

The compression ratio of the ACW(n, s) scheme can be expressed as

$$C = \frac{S_o}{S_c} = \frac{S_o}{16 + S_F + s(8+L) + \lceil N/n \rceil}. \tag{10}$$

By (10), C mainly depends on the values of s and n.In this work, the compression algorithm starts by selecting a range of n values, and then for each value of n, C is calculated. The character wordlength selected to compress the source file is the one that achieves the highest compression ratio.

5 Experimental results

In order to evaluate its performance, the ACW(n, s) scheme is implemented using C++ language and used to compress text files from standard corpora such as Calgary, Canterbury, and Large Corpora[13,20].

At this stage, little effort has been taken to optimize the runtime of the compression-decompression prototype codes; therefore, only results obtained for the compression ratio are presented.

In this paper, five experiments are performed and presented to evaluate and compare the performances of the ACW(n,s) scheme.

Experiment 1. This experiment investigates the variation of C and s with n using two coding formats, namely, the ASCII and the adaptive coding formats. The compression ratios achieved for three text files of various sizes from the Calgary corpus (bib, book1, and paper2), are given. The results are shown in Table 4.

Table 4 Variation of C and s with n using two coding formats for the ACW(n, s) scheme

n	bib				book1				paper2			
	ASCII coding		Adaptive coding		ASCII coding		Adaptive coding		ASCII coding		Adaptive coding	
	s	C	s	C	s	C	s	C	s	C	s	C
9	176	1.262	82	1.275	1056	1.266	237	1.281	113	1.265	28	1.280
10	214	1.394	140	1.406	1352	1.397	600	1.414	144	1.397	68	1.413
11	223	1.523	174	1.537	1447	1.531	865	1.547	155	1.530	95	1.546
12	220	0.813	190	0.930	1448	0.898	1033	1.164	155	0.846	113	1.092
13	211	0.701	194	0.813	1412	0.792	1135	1.064	151	0.745	123	0.997
14	111	1.470	111	1.477	552	1.647	552	1.674	54	1.637	54	1.641
15	191	0.545	185	0.632	1296	0.634	1185	0.853	138	0.599	127	0.784
16	181	0.499	178	0.570	1233	0.573	1162	0.763	132	0.544	124	0.701

The results shows that: 1) a compression of over 50% can be achieved, 2) the maximum compression is achieved at $n \leqslant 14$ depending on the size and the contents of the source file, and 3) the adaptive coding yields higher compression ratios than the ASCII coding for all values of n.

Experiment 2. This experiment investigates the effect of the character coding formats, namely, the ASCII coding, Huffman coding, and the adaptive coding, on the compression ratio achieved by the ACW(n,s) scheme. The results are listed in Table 5. The results reveal that the compression ratio achieved using adaptive coding is higher than when using ASCII coding by a very small margin (around 2%). However, the ratio achieved using adaptive coding is smaller than that using Huffman coding. This is because Huffman coding itself is a compression algorithm, which means two compression techniques (Huffman coding and ACW(n, s)) are sequentially applied on the text. First, the text is compressed during the coding process using Huffman coding. Second, the resultant binary sequence is compressed using the ACW(n, s) scheme. So, the resultant compression ratio is the product of the compression ratios of Huffman and the ACW(n,s) scheme.

Table 5 shows that the optimum character wordlength for the ASCII and the adaptive coding are either 11 or 14 depending on the context and size of the text file. However, it is difficult to make early prediction to the optimum value, but still it can be easily recognized that the optimum value can be reached after a small number of iteration. On the other hand, the compression ratio for Huffman coding is followed by the ACW(n, s) for all examined text files, and it is achieved when $n = 11$.

Table 6 lists the compression ratios achieved by Huffman coding[21,22], the ACW(n, s) scheme, and the total compression ratio achieved by cascading the two techniques. It shows that the new scheme achieves almost a fix compression ratio advantage of about 1.5 over Huffman coding, which is less than what is achieved over adaptive coding. For example, for the bib text file, the compression ratios achieved by the ACW(n,s) scheme using the ASCII, Huffman, and the adaptive coding formats are 1.528, 1.525, and 1.537, respectively.

Experiment 3. This experiment investigates the scheme compression efficiency (E_n). Text files are usually coded using 7-bit character wordlength; therefore, using n-bit character wordlength should provide a compression ratio of $n/7$. For example, for 14-bit character wordlength, we should ideally achieve a compression ratio of 2 simply by replacing each 14-bit with its equivalent character. The compression ratio obtained through dividing the character wordlength by 7 is referred to as the ideal compression ratio and it is denoted by C_{ideal}.

Table 5 Comparison of C achieved by the ACW(n, s) scheme using different coding formats

Corpus	Filename	Size (byte)	ASCII/ACW(n, s)		Huffman/ACW(n, s)		Adaptive/ACW(n, s)	
			C	n	C	n	C	n
Calgary	bib	111261	1.528	11	2.330	11	1.537	11
	book1	768771	1.647	14	2.673	11	1.674	14
	book2	610856	1.531	14	2.530	11	1.545	11
	paper1	53161	1.529	11	2.431	11	1.542	11
	paper2	82199	1.637	14	2.630	11	1.641	14
	paper3	46526	1.611	14	2.595	11	1.616	14
	paper4	13286	1.615	14	2.567	11	1.616	14
	paper5	11954	1.526	11	2.432	11	1.539	11
	paper6	38105	1.549	14	2.416	11	1.551	14
Canterbury	alice29.txt	152089	1.650	14	2.643	11	1.656	14
	asyoulik.txt	125179	1.646	14	2.516	11	1.648	14
	lcet10.txt	426754	1.592	14	2.599	11	1.604	14
	plrabn12.txt	481861	1.743	14	2.667	11	1.750	14
Large	bible.txt	4047392	1.841	14	2.783	11	1.842	14
	world192.txt	2473400	1.537	14	2.420	11	1.549	14

Table 6 Comparison of C for Huffman coding, ACW(n, s) scheme, and a combination of these two techniques

Corpus	Filename	Size (byte)	Huffman coding	ACW(n, s) scheme		Huffman/ACW(n, s)	
				C	n	C	n
Calgary	bib	111261	1.529	1.524	11	2.330	11
	book1	768771	1.754	1.524	11	2.673	11
	book2	610856	1.659	1.525	11	2.530	11
	paper1	53161	1.595	1.524	11	2.431	11
	paper2	82199	1.726	1.524	11	2.630	11
	paper3	46526	1.705	1.522	11	2.595	11
	paper4	13286	1.690	1.519	11	2.567	11
	paper5	11954	1.601	1.519	11	2.432	11
	paper6	38105	1.586	1.524	11	2.416	11
Canterbury	alice29.txt	152089	1.734	1.524	11	2.643	11
	asyoulik.txt	125179	1.651	1.524	11	2.516	11
	lcet10.txt	426754	1.703	1.526	11	2.599	11
	plrabn12.txt	481861	1.749	1.525	11	2.667	11
Large	bible.txt	4047392	1.824	1.526	11	2.783	11
	world192.txt	2473400	1.587	1.525	11	2.420	11

It is possible to evaluate the efficiency of the new scheme by comparing the actual and the ideal compression ratios. Thus, a new parameter is defined, namely, the scheme efficiency (E_n), which is a function of n. It is calculated as the ratio between the actual and the ideal compression ratios at a particular character wordlength (n).

The efficiencies of the ACW(n,s) scheme are given in Table 7. The results demonstrate that although a higher compression ratio is achieved using a bigger character wordlength, at the same time the scheme efficiency is less. For example, E_n is around 97% for 11-bit character wordlength and nearly 80% for 14-bit character wordlength. The adaptive coding achieves higher efficiencies than both ASCII and Huffman coding formats.

Experiment 4. In this experiment, the compression ratio achieved by the ACW(n,s) scheme is compared with other data compression algorithms, such as Huffman coding (HU), fixed-length Hamming (FLH), Huffman followed by fixed-length Hamming (HF), and the HCDC(k) scheme. The results for the ACW(n, s) scheme, which are based on Huffman and adaptive coding formats, are presented in Table 8 for two text files from the Calgary corpus, namely, book1 and paper1. Results for HU, FLH, and HF are from [17], and results for the HCDC(k) scheme are from [6].

It is clear that Huffman coding followed by the ACW(n,s) scheme gives the highest compression ratio for both files, and it is higher than Huffman coding followed by fixed-length Hamming algorithm. However, the compression ratio based on adaptive coding provides nearly the same performance as the other algorithms.

Table 9 compares the compression ratio of the new ACW(n,s) scheme with a number of adaptive schemes as follows: the Unix compact utility based on adaptive Huffman (AH), the greedy adaptive Fano coding (AF)[23], and the HCDC(k) scheme[6]. The compression ratio of the ACW(n,s) scheme based on Huffman coding is higher than all of other schemes for all text files examined. However, using the adaptive coding provides a compression ratio that is very close to the three algorithms with which it is compared.

Table 7 Comparison of C and E_n of ACW(n,s) scheme for different coding formats

Corpus	Filename	ASCII/ACW(n,s)		Huffman/ACW(n,s)		Adaptive/ACW(n,s)	
		C (n)	E_n	C (n)	E_n	C (n)	E_n
Calgary	bib	1.528 (11)	97.2	1.524 (11)	97.0	1.537 (11)	97.8
	book1	1.647 (14)	82.4	1.524 (11)	97.0	1.674 (14)	83.7
	book2	1.531 (11)	97.4	1.525 (11)	97.1	1.545 (11)	98.3
	paper1	1.529 (11)	97.3	1.524 (11)	97.0	1.542 (11)	98.1
	paper2	1.637 (14)	91.9	1.524 (11)	97.0	1.641 (14)	82.1
	paper3	1.611 (14)	80.5	1.522 (11)	96.9	1.616 (14)	80.8
	paper4	1.615 (14)	80.8	1.519 (11)	96.7	1.616 (14)	80.8
	paper5	1.526 (11)	97.1	1.519 (11)	96.7	1.539 (11)	97.9
	paper6	1.549 (14)	77.5	1.524 (11)	97.0	1.551 (14)	77.6
Canterbury	alice29.txt	1.650 (14)	82.5	1.524 (11)	97.0	1.656 (14)	82.8
	asyoulik.txt	1.646 (14)	82.3	1.524 (11)	97.0	1.648 (14)	82.4
	lcet10.txt	1.592 (14)	79.6	1.526 (11)	97.1	1.604 (14)	80.2
	plrabn12.txt	1.743 (14)	87.2	1.525 (11)	97.1	1.750 (14)	87.5

Experiment 5. In this experiment, the performance of the ACW(n, s) scheme in terms of coding rate (C_r) in bit-per-character (bpc) is compared with a number of widely used programs and state-of-the-art software[6,18,24]. The results are tabulated in Table 10. The coding rate of the new scheme is found to be competing with many standard softwares, and it is better if the ACW(n, s) uses Huffman coding in converting text into binary sequence.

Table 8 Comparison of C of the ACW(n, s) scheme with various compression algorithms

Algorithm	book1	paper1
Huffman coding (HU)	1.724	1.595
Fixed-length Hamming (FLH)	1.143	1.143
HU following FLH (HF)	1.707	1.565
HCDC(k)	2.543 (6)*	1.895 (4)*
Adaptive/ACW(n,s)	1.674 (14)$^+$	1.542 (11)$^+$
Huffman/ACW(n,s)	2.673 (11)$^+$	2.431 (11)$^+$

* Represents k, which is the number of repetition loops. $^+$ Represents n, which is the adaptive character wordlength.

Table 9 Comparison of C of ACW(n,s) scheme and various adaptive compression algorithms

Corpus	Filename	AH[23]	AF[23]	HCDC(k) scheme[6]	ACW(n,s) scheme	
					Adaptive	Huffman
Calgary	bib	1.526	1.524	1.632 (4)	1.537 (11)	2.330 (11)
	book1	1.753	1.750	2.543 (6)	1.674 (14)	2.673 (11)
	book2	1.658	1.653	2.253 (5)	1.545 (11)	2.530 (11)
	paper1	1.587	1.588	1.895 (4)	1.542 (11)	2.431 (11)
Canterbury	alice29.txt	1.753	1.746	2.397 (5)	1.656 (14)	2.643 (11)
	asyoulik.txt	1.648	1.645	2.086 (5)	1.648 (14)	2.516 (11)
	lcet10.txt	1.718	1.717	2.296 (5)	1.604 (14)	2.599 (11)
	plrabn12.txt	1.769	1.766	2.554 (6)	1.750 (14)	2.667 (11)

Table 10 Comparison of the C_r of the ACW(n, s) with various standard programs and state-of-the-art software

Programs	Type	Version	Filename					
			bib	book1	book2	paper1	paper2	alice29
compress	LZ	4.0	3.350	3.460	3.280	3.770	3.520	-
gzip	LZ	1.2.3	2.520	3.260	2.710	2.800	2.900	-
comp-2-o-4	PPM	Trial V.	2.020	2.350	2.080	2.480	2.450	-
DD	-	Trail V.	2.530	2.690	2.390	3.080	2.850	-
compress	LZ	4.3d	-	3.486	-	-	-	3.270
pkzip	LZ	2.04e	-	3.288	-	-	-	2.884
gzip-9	LZ	1.2.4	-	3.250	-	-	-	2.848
szip-b41-o0	BWT	1.05Xf	-	2.345	-	-	-	2.239
ha a2	PPM	0.98	-	2.453	-	-	-	2.171
boa-m15	PPM	0.58b	-	2.204	-	-	-	2.061
rkive-mt3	PPM	1.91b1	-	2.120	-	-	-	2.055
neural small	NN	P5	-	2.508	-	-	-	2.301
neural large	NN	P6	-	2.283	-	-	-	2.129
HCDC(k)	Bit-level	Trial V.	4.289(4)*	2.753(6)*	3.107(5)*	3.694(4)*	2.809(6)*	2.920(5)*
ACW(n,s) (adaptive)	Bit-level	Trial V.	4.554(11)$^+$	4.182(14)$^+$	4.531(11)$^+$	4.540(11)$^+$	4.266(14)$^+$	4.227(14)$^+$
ACW(n,s) (Huffman)	Bit-level	Trial V.	3.004(11)$^+$	2.619(11)$^+$	2.767(11)$^+$	2.880(11)$^+$	2.662(11)$^+$	2.649(11)$^+$

* Represents k, which is the number of the repetition loops. $^+$ Represents n, which is the adaptive character wordlength.

6 Conclusions

The main conclusions of this work can be summarized as follows:

1) The new scheme prevails is superior to all other schemes that degrade the performance of the ACW(n) algorithm.

2) The new scheme has an excellent, comparable, and competitive performance and outperforms the widely used data compression algorithms and state-of-the-art compression tools of different models.

3) The compression ratio depends on i) the adaptive character wordlength used and the number of subsequences into which the binary sequence is subdivided, ii) the size of the source file, and the frequencies and distribution of characters within the file, and iii) text-to-binary coding format that is used to convert a text file into a binary sequence.

4) The maximum compression ratio is always achieved when n varies between 9 and 14 bits. This of course eliminates the needs of finding the optimum n by lengthy search. In worst case, the search for optimum n is to be performed over a small range, for example, from 9 to 16.

5) The three coding formats we investigated (ASCII, Huffman, and adaptive coding) reveal that the highest compression ratio achieved is for Huffman coding since it performs coding and compression at the same time.

6) The new scheme can be used as a complementary scheme to any statistical lossless compression algorithm, such as Shanon-Fano coding, static or adaptive Huffman coding, arithmetic coding, the combination of these algorithms, or any modified form of them.

7) The results on E_n demonstrated an efficiency of around 97% using 11-bit character wordlength and around 80% using 14-bit character wordlength.

   Our future work will focus on the performance of this scheme in compressing multimedia files, and comparison of the compression/decompression processing time with other compression algorithms and state-of-the-art compression tools.

References

[1] J. Lánský, M. Žemlička. Text compression: Syllables. In *Proceedings of the Dateso Workshop on Databases, Texts, Specifications and Objects*, pp. 32–45, 2005.

[2] A. Mofat, R. Y. K. Isal. Word-based text compression using the burrows-wheeler transform. *Information Processing and Management*, vol. 41, no. 5, pp. 1175–1192, 2005.

[3] J. Adiego, P. de la Feunte. On the use of words as source alphabet symbols in PPM. In *Proceedings of Data Compression Conference*, IEEE, pp. 435, 2006.

[4] J. Dvorsky, J. Pokorny, V. Snasel. Word-based compression methods for large text documents. In *Proceedings of Data Compression Conference*, IEEE, pp. 523, 1999.

[5] J. Lánský, M. Žemlička. Compression of a dictionary. In *Proceedings of DATESO Workshop on Databases, Texts, Specifications and Objects*, pp. 11–20, 2006.

[6] H. Al-Bahadili, A. Rababa′a. An adaptive bit-level text compression scheme based on the HCDC algorithm. In *Proceedings of Mosharaka International Conference on Communications, Networking and Information Technology*, Amman, Jordan, pp. 51–56, 2007.

[7] H. Al-Bahadili, S. M. Hussain. An adaptive character wordlength algorithm for data compression. *Computers & Mathematics with Applications*, vol. 55, no. 6, pp. 1250–1256, 2008.

[8] Y. Weng, J. Jiang. Real-time and automatic close-up retrieval from compressed videos. *International Journal of Automation and Computing*, vol. 5, no. 2, pp. 198–201, 2008.

[9] L. Zhu, G. Y. Wang, C. Wang. Formal photograph compression algorithm based on object segmentation. *International Journal of Automation and Computing*, vol. 5, no. 3, pp. 276–283, 2008.

[10] K. Saydood. *Introduction to Data Compression*, 3rd ed., Morgan Kaufmann, 2006.

[11] Y. Ye, P. Cosman. Dictionary design for text image compression with JBIG2. *IEEE Transactions on Image Processing*, vol. 10, no. 6, pp. 818–828, 2001.

[12] I. H. Witten, A. Moffat, T. C. Bell. Managing gigabytes: Compressing and indexing documents and images. *IEEE Transactions on Information Theory*, vol. 41, no. 6, Part 2, pp. 2101–2102, 1995.

[13] T. C. Bell, J. G. Cleary, I. H. Witten. *Text Compression*, NJ, USA: Prentice-Hall, 1990.

[14] H. Al-Bahadili. A novel lossless data compression scheme based on the error correcting Hamming codes. *Computers & Mathematics with Applications*, vol. 56, no. 1, pp. 143–150, 2008.

[15] S. Nofal. Bit-level text compression. In *Proceedings of the 1st International Conference on Digital Communications and Computer Applications*, Irbid, Jordan, pp. 486–488, 2007.

[16] G. Caire, S. Shamai, S. Verdu. Noiseless data compression with low density parity check codes. *Advances in Network Information Theory, DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, P. Gupta, G. Kramer, A. J. van Wijngaarden, Ed., vol. 66, pp. 263–284, 2004.

[17] A. A. Sharieh. An enhancement of Huffman coding for the compression of multimedia files. *Transactions of Engineering Computing and Technology*, vol. 3, no. 1, pp. 303–305, 2004.

[18] M. V. Mahoney. Fast text compression with neural networks. In *Proceedings of the 13th International Florida Artificial Intelligence Research Society Conference*, pp. 230–234, 2000.

[19] A. Rababa′a. An Adaptive Bit-Level Text Compression Scheme Based on the HCDC Algorithm, M. Sc. dissertation, Amman Arab University for Graduate Studies, Amman, Jordan, 2008.

[20] R. Arnold, T. Bell. A corpus for the evaluation of lossless compression algorithms. In *Proceedings of the Conference on Data Compression*, IEEE, pp. 201–210, 1997.

[21] J. S. Vitter. Dynamic Huffman codes. *Journal of the ACM*, vol. 34, no. 4, pp. 158–167, 1989.

[22] J. S. Vitter. Design and analysis of dynamic Huffman coding. *Journal of the ACM*, vol. 34, no. 4, pp. 825–845, 1987.

[23] L. Rueda, B. J. Oommen. A fast and efficient nearly-optimal adaptive Fano coding scheme. *Information Science*, vol. 176, no. 12, pp. 1656–1683, 2006.

[24] H. Plantinga. An asymmetric, semi-adaptive text compression algorithm. In *Proceedings of IEEE Data Compression Conference*, 1994.

Hussein Al-Bahadili received the B. Sc. degree in engineering from University of Baghdad, Iraq in 1986, and the M.Sc. and Ph.D. degrees in engineering from University of London, UK in 1988 and 1991, respectively. He is currently an associate professor at the Arab Academy for Banking and Financial Sciences (AABFS). He is a visiting researcher at the Wireless Networks and Communications Centre (WNCC) at University of Brunel, UK. He is also a visiting researcher at the Centre of Osmosis Research and Applications (CORA), University of Surrey, UK.

His research interests include parallel and distributed computing, wireless communications, computer networks, cryptography and network security, data compression, image processing, and artificial intelligence and expert systems.

E-mail: hbahadili@aabfs.org (Corresponding author)

Shakir M. Hussain received the B. A. degree in statistics from University of Al-Mustansiriyah, Iraq in 1976 and M. Sc. degree in computing and information science from Oklahoma State University, USA in 1984. In 1997, he received the Ph.D. degree in computer science from University of Technology, Iraq. From 1997 to 2008, he was a faculty member at Applied Science University, Jordan. Currently, he is the head of Computer Science Department at Petra University, Jordan. He is a member of ACM.

His research interests include block cipher, key generation, authentication, and data compression.

E-mail: shussein@uop.edu.jo