An Effective Method For Data Compression Based On Adaptive Character Wordlength

Wiam Al Hayek
Department of Social and Applied Sciences, Prince Alia University College,
Al Balqa Applied University, Jordan

Abstract: *The adaptive character wordlength (ACW) algorithm is a bit-level, lossless, adaptive, and asymmetric text compression algorithm that is recently proposed. In this algorithm, the binary sequence is divided into a number of blocks (B) each of n-bit length (n > 8). This gives each block a possible decimal values ranges from 0 to 2^n-1. If the number of the different decimal values (d) is equal to or less than 256 (d ≤ 256), then the binary sequence can be compressed using n-bit character wordlength, rather than using the standard 8-bit character wordlength. Thus, a compression ratio of approximately n/8 can be achieved. Since the compression ratio is a function of n, this algorithm is referred to as ACW(n).*

Keywords: *Data Compression, Algorithm, Scheme.*

Received October 20, 2010; accepted February 26, 2012

1. Introduction

Data compression algorithms are designed to reduce the size of data so that it requires less disk space for storage and less bandwidth to be transmitted over data communication channels of limited bandwidth [11]. An additional benefit of data compression is that it decreases the amount of errors during data transmission over error-prune communication channels, by decreasing the size of information to be exchanged over such channels [1, 6, 10].

Data compression is usually obtained by substituting a shorter symbol for an original symbol in the source data, containing the same information but with a smaller representation in length. The symbols may be characters, words, phrases, or any other unit that may be stored in a dictionary of symbols and processed by a computing system [9, 15].

Data compression requires efficient algorithmic transformations of data representations to produce more compact representations. Such algorithms are known as data compression algorithms or data encoding algorithms. Each data compression algorithm needs to be complemented by its inverse, which is known as a data decompression algorithm (or also can be referred to as a data decoding algorithm), to restore an exact or an approximate form of the original data [2, 7].

The basic data compression/ decompression block diagram is illustrated in figure 1. It is shown as a black box, in which the original data stream may be processed according to one or more compression algorithms, which generate a compressed data stream. On the other hand, the compressed data stream may be processed according to one or more decompression algorithms to reproduce exact/approximate original data stream [7].

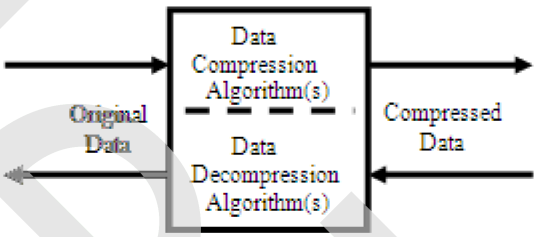

Figure 1. Basic data compression/decompression block diagram.

Data coding techniques have been widely used in developing data compression algorithms, since coding techniques may lend themselves well to the above concept [10].

Data coding involves processing an input sequence:

$$\boldsymbol{X} = \{x[1],\ x[2],\ldots\ x[M]\} \quad (1)$$

Where each input symbol, $x[i]$, is drawn from a source alphabet:

$$\boldsymbol{S} = \{s_i,\ s_i\ \ldots,\ s_m\} \quad (2)$$

Whose probabilities are:

$$\boldsymbol{\rho} = \{\rho_1,\ \rho_2,\ \ldots,\ \rho_m\} \text{ with } 2 \le m < \infty \quad (3)$$

For example, for binary alphabet $m = 2$, $\boldsymbol{S}$ is either 0 or 1. The encoding process is rendered by transforming $\boldsymbol{X}$ into an output sequence:

$$\boldsymbol{Y} = \{y[1], y[2], \ldots, y[Q]\} \quad (4)$$

Where each output symbol $y[i]$ is drawn from a code alphabet:

$$\boldsymbol{A} = \{a_1, a_2\ldots, a_q\} \quad (5)$$

Here the code alphabet is still binary (i.e., either 0 or 1, $q = 2$), but Q must be much less than M. The main problem in data compression is to find an encoding scheme that minimizes the size of ***Y***, in such a way that ***X*** can be completely recovered by applying the scheme that minimizes the size of ***Y***, in such a way that ***X*** can be completely recovered by applying the decompression process

2. Adaptive Character Wordlength

Our proposed method can be classified as a bit-level, lossless, adaptive, and asymmetric data compression algorithm [2, 15]. In this algorithm, the data symbols (characters) of a source file are converted to a binary sequence by concatenated the individual binary codes of the data symbols.

The binary sequence is, then, subdivided into a number of blocks, each of n-bit length ($n > 8$). The last block is padded with 0s if the remaining bits are less than n. For a binary sequence of N-bit length, the number of blocks B (where B is a positive integer number) is given by:

$$B = \left\lceil \frac{N}{n} \right\rceil \quad (6)$$

The number of padding bits (g), which is added to the last block is calculated by:

$$g = B * n - N \quad (7)$$

For these B blocks, the equivalent decimal value of each block lies between 0 to 2^n-1. The number of the different decimal values (d), these blocks may have, is between 1 to 2^n. That is because not all possible n-bit binary sequences could exist, and some blocks may have the same decimal values (binary sequence). Practically, 1 represents that all blocks have the same decimal value, and 2^n represents that all possible decimal values are assigned to different blocks. Although, $B > 2^n$, it is still possible that $d < 2^n$, which means that not all possible decimal values are used and some blocks have the same value.

In order to be able to compress a binary sequence using n-bit character wordlength, d should be less than or equal to 256 ($d \leq 256$). If this condition is satisfied (i.e., $d \leq 256$), then these d values are arranged in ascending order, and each block is converted to symbol (character) according to its sequence number, and not according to its actual decimal value. In this case, it is clear that B is equivalent to the number of characters that will be written to the compressed file.

Otherwise, if the above condition can be fulfilled (i.e., $d > 256$), then the file can not be compressed using n-bit character wordlength. Thus, another character wordlength should be examined, until a particular value is found for n or the source file can not be compressed using more than 8-bit character wordlength. Therefore, this algorithm is referred to as ACW(n).

The ACW(n) algorithm can be used to compress a binary sequence of data regardless of the way that has been adopted to create that particular binary sequence. Therefore, it can also be used as complementary compression algorithm for other data compression algorithms. For example, it can be used as a complementary algorithm to statistical compression algorithms, in such algorithms, the characters in the source file are converted to a binary sequence, where the most common characters in the file have the shortest binary codes, and the least common have the longest, the binary codes are generated based on the estimated probability of the character within the file. Then, instead of compressing the generated binary sequence using 8-bit character wordlength, as it is usually done, the ACW(n) algorithm can be used to reduce the size of the compressed file, thereafter, enhances the compression process. The ACW(n) algorithm, in such cases, searches for the maximum possible character wordlength n ($n > 8$), so that the compression ratio can improved and approximately increased by a factor of n/8.

3. Algorithm For Adaptive Character Wordlength

The steps of the ACW(n) algorithm can be summarized as follows:

1. The binary sequence is divided into blocks of an n-bit length. If the length of the binary sequence is not a multiple of n, then the last block would have less than n bits, so that it is padded with 0s. The equivalent decimal value of each block is between 0 to 2^n-1. In total, there are 2^n possible values.
2. Calculate the number of the different values the blocks may have (d), and the frequency or the probability of occurrence of each block (f).
3. If the number of the different decimal values (d) is equal to or less than 256 ($d \leq 256$), then a n-bit character wordlength can be used to convert or map the blocks into character as follows:
    a. Sort the blocks descendingly according to their frequencies. So that the most common block takes a 0 sequence number, and the least common block takes a d-1 sequence number.
    b. Convert each block to character according to its sequence number, which in this case, lies between 0 and d-1, and not according to its equivalent decimal value.
    c. Construct the header of the compressed data file. The header includes the information, which is necessary to restore the original binary sequence during the decompression process. This information, for example, includes: (i) the character wordlength (n), (ii)

the number of padded bits (g), (iii) the number of the different decimal values the blocks may have (d), and (iv)

d. The equivalent decimal values of the sorted blocks.

4. If d is more than 256, then a different value of n should the examined. This process is continued until; if possible, the above condition is satisfied. If no value of n can be found that satisfies the condition of $d \leq 256$, then the binary sequence can only be converted to character sequence using an 8-bit character wordlength.

4. The Adaptive Character Coding Format

The coding format that is used in converting a source file to a binary sequence has an enormous effect on the binary sequence entropy. Thereafter, it affects the bpc required to represent characters in the compressed file or the compression ratio that can be achieved.

A conventional coding format is the ASCII coding, in which each character within the text file is represented by 7-bit character wordlength. The length of the binary sequence in bits is given by:

$$N_{ASCII} = 7S_o \tag{8}$$

Another coding format is the Huffman coding. Using Huffman coding, the length of the binary sequence may be expressed as:

$$N_{Huff} = S_o \sum_{i=1}^{N_c} f_i \, w_i \tag{9}$$

Where N_c is the number of symbols (character types) within the source file.

f_i is the frequency or the probability of occurrence of the i^{th} character,

w_i is the number of bits representing the i^{th} character.

S_o is the number of characters within the source file (or size of the file in Bytes).

In this paper, a new coding format is introduced and investigated, namely, the adaptive coding format. In adaptive coding, first, the character frequencies are calculated and sorted in ascending order from the most common character to the least, similar to Huffman coding. Second, the most common character is given a 0 sequence number, while the least common character is given N_c-1 sequence number. Then, each character is coded to binary according to its sequence number. For example, the equivalent binary codes for the most (first), second, and the third characters are 0000000, 0000001, and 0000010, respectively.

This form of coding ensures a low entropy binary sequence, therefore, we expect a higher compression ratio and lower bpc is required to represent characters within the compressed file. In order not to get the data mixed up during the decompression phase, the number of the sorted characters and the characters themselves should be included in the compressed file header. This of course will add an overhead of not more than 129 bytes. It is clear that this overhead is small as compared to the size of the data file.

5. The Adaptive Character Wordlength (Acw(N,S)) Scheme

A new scheme that is developed to enhance the compression ratio of the original ACW(n) algorithm, and eliminate all drawbacks that may degrade the performance of the algorithm. In this new scheme, the binary sequence is subdivided into a number of subsequences (s) of variable sizes, each of them satisfies the condition that $d \leq 256$, i.e., each subsequence enfolds a number of blocks that have less than or equal to 256 different decimal values. Therefore, the new scheme is referred to as ACW(n,s). Furthermore, each of the sequences will have it own header that encloses the number of block, and the number of different values (d) within this subsequence. It should be noted that d is equal to 256 for all subsequences, except the last one where it may be less than 256. This adds an extra overhead that directly proportional to s.

Subdividing the original binary sequence into subsequences eliminates the first downside of $d \leq 256$ and consequently the agonize of having low success probability if large values of n are used. But still, we can not use large values of n, because as n increases, the size of the header of each subsequence is also increased. Thus, an optimization mechanism is required to find the optimum character wordlength.

In this adaptive coding format, an original character is encoded to binary according to its frequency. This encoding reduces the entropy of the binary sequence so it grants higher compression ratios. However, the type of coding used should be indicated in the compressed file header. The ACW(n,s) algorithm can be used to compress a sequence of binary data regardless of the way that has been adopted to create the binary sequence. Therefore, it can also be used as a complementary compression algorithm for other data compression algorithms. In particular, it can be used as a complementary algorithm to statistical compression techniques, where in such techniques, the characters in the source file are converted to a binary code, where the most common characters in the file have the shortest binary codes, and the least common have the longest, the binary codes are generated based on the estimated probability of the character within the file.

Then, instead of compressing the generated binary stream using 8-bit character wordlength, as it is usually done, the ACW(n,s) algorithm can be used to further reduce the size of the compressed file, thereafter, enhances the compression process. The ACW algorithm, in such cases, searches for the maximum

possible character wordlength n ($n > 8$), so that the compression ratio can improved and approximately increased by a factor of $n/8$ as it has been explained above.

Storing this information in the compressed file is very useful in two terms, first it ensures an accurate or exact retrieval to the original data, and second it enables a fast decompression as compared to the compression process.

6. Analyzing The Performance Of The Acw(N,S) Scheme

The compression ratio that can be achieved using the ACW(n,s) can be expressed as:

$$C = \frac{S_o}{S_c} = \frac{S_o}{S_H + B} = \frac{S_o}{16 + S_F + s(8+L) + \frac{N}{n}} \quad (10)$$

The above equation is a nonlinear equation, and C is a function of n and s. As n increases, S_H and B behaves differently. While B decreases, S_H increases. Therefore, there must be an optimum value for n where C would have its maximum value. Due to the nonlinear nature of the above equation it is difficult to be solved analytically, and only iterative solution can be used to find the optimum value of n.

Table (2)

7. Experiment Result And Discussion

In order to evaluate the performance of the ACW(n,s) scheme, it is used to compress a number of text files from standard corpora, namely, Calgary corpus, Canterbury corpus, Artificial corpus, Large corpus, and Miscellaneous corpus [4, 5, 16]. At this stage, little effort has been taken to optimize the runtime of the compression-decompression prototype code, therefore, in this paper, we only compare and show the results for the compression ratio.

7.1. Comparison Of The Compression Ratio Of The Acw(N,S) Scheme With Other Data Compression Algorithms.

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 error correcting Hamming code data compression (HCDC(k)) scheme. The results presented are for text file from the Calgary corpus (book1, and paper1).

The results for the ACW(n,s) scheme are based on Huffman and adaptive coding formats. 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. But the compression ratio based on adaptive coding provides nearly the same performance as the other algorithms.

Table 1. Comparison of the compression ratio (C) between the ACW(n,s) scheme and various compression algorithms.

Algorithm	book1	paper1
HU[1]	1.724	1.595
FLH[1]	1.143	1.143
HF[1]	1.707	1.565
HCDC(k)[2]	2.543 (6)	1.895 (4)
ACW(n,s) – Adaptive coding	1.674 (14)	1.542 (11)
ACW(n,s) – Huffman coding	2.673 (11)	2.431 (11)

HU: Huffman coding.
FLH: Fixed-length Hamming.
HF: HU following FLH.
HCDC(k): The error correcting Hamming code data compression scheme.
[1] Results for HU , FLH and HF are from [13].
[2] Results for the HCDC(k) scheme are from [3].

Table 2 compares the compression ratio of the new ACW(n,s) scheme with two adaptive schemes, namely, the Unix compact utility that is based on adaptive Huffman (AH) and the greedy adaptive Fano coding (AF) for more text files [12]. The compression ratio of the ACW(n,s) scheme based on Huffman coding is higher than Unix compact utility, greedy adaptive Fano coding, and error correcting Hamming code data compression (HCDC(k)) scheme for all text files examined. However, using the adaptive coding provides a compression ratio that is very close the three algorithms it is compared with.

Table 2. Comparison of the compression ratio (C) of ACW(n,s) scheme and various adaptive compression algorithms.

	File Name	AH[1]	AF[1]	HCDC(k)[2]	ACW(n,s) scheme	
					Adaptive	Huffman
Calgary corpus	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 corpus	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)

AH: Unix compact utility.
AF: Greedy adaptive Fano coding.
HCDC(k): The error correcting Hamming Code Data Compression scheme.
[1] Results for AH , AF are from [14].
[2] Results for the HCDC(k) scheme are from [3].

Table 3. Comparison of the compression ratio (C) of ACW(n,s) scheme for different coding formats.

Corpus	File Name	ASCII coding	Huffman coding	Adaptive coding
		C (n)	C (n)	C (n)
Calgary corpus	Bib	1.528 (11)	1.524 (11)	1.537 (11)
	book1	1.647 (14)	1.524 (11)	1.674 (14)
	book2	1.531 (11)	1.525 (11)	1.545 (11)
	paper1	1.529 (11)	1.524 (11)	1.542 (11)
	paper2	1.637 (14)	1.524 (11)	1.641 (14)
	paper3	1.611 (14)	1.522 (11)	1.616 (14)
	paper4	1.615 (14)	1.519 (11)	1.616 (14)
	paper5	1.526 (11)	1.519 (11)	1.539 (11)
	paper6	1.549 (14)	1.524 (11)	1.551 (14)
Canterbury corpus	alice29.txt	1.650 (14)	1.524 (11)	1.656 (14)
	asyoulik.txt	1.646 (14)	1.524 (11)	1.648 (14)
	lcet10.txt	1.592 (14)	1.526 (11)	1.604 (14)
	plrabn12.txt	1.743 (14)	1.525 (11)	1.750 (14)

Table 3 obtained the result for the compression ratio of the ACW(n,s) scheme for different coding format (ASCII coding, Huffman coding, Adaptive coding) for two types of standard corpa (Calgary corpus and Canterbury corpus). The results demonstrate that a higher compression ratio is achieved using Adaptive character wordlength.

8. Conclusion

This paper presents a description and performance evaluation of new bit-level, lossless, adaptive, and asymmetric data compression scheme, namely, the ACW(n,s) scheme. It utilizes the adaptive character wordlength (ACW(n)) algorithm. n and s refer to the character wordlength and the number of subsequences into which the original binary sequence is subdivided into, respectively. In addition, in this paper, in order to further enhance the performance of the ACW(n,s) algorithm, an adaptive text-to-binary coding format is developed. In this coding format, an uncompressed character is coded to binary according to its frequency, rather than its equivalent ASCII code. Thus, a higher compression was achieved, since this coding format reduced the entropy of the generated binary sequence.

References

[1] Adiego J., G. Navarro and et. , "*Using Structural Contexts to Compress Semi-Structured Text Collections*", *Information Processing and Management*, Vol. 43, Issue 3, pp. 769–790, 2007.

[2] Al-Bahadili H. and et., "*An Adaptive Character Wordlength Algorithm for Data Compression*", *Journal of Computers & Mathematics with Applications*, Vol. 55,Issue 6, pp.1250-1256, March 2008.

[3] Al-Bahadili H. and et., "*An Adaptive Bit-Level Text Compression Scheme Based on the HCDC Algorithm*", *Proceedings of Mosharaka International-Conference on Communications Networking and Information Technology (MIC-CNIT 2007)*, Amman-Jordan, 6-8 December, 2007.

[4] Bell T. and et., "*Text Compression", Prentice-Hall Adv. Ref. Series: Computer Science*, 1990.

[5] Bell T. and et. , "*Modeling for Text Compression*", *ACM Computing Surveys*, Vol. 21, No.4, pp. 557-591, 1989.

[6] Freschi V. and et. , "*Longest Common Subsequence Between Run-Length-Encoded String: a New Algorithm with Improved Parallelism*", *Information Processing letters*, Vol. 90, pp. 167-173, 2004.

[7] Gryder R. and et. , "*Survey of Data Compression Techniques*", *ORNL/TM-* 11797, 1991.

[8] Lansky J. and et. , "*Compression of a Dictionary*", *Proceedings of the Dateso 2006 Annual International Workshop on Databases, Texts*, Specifications and Objects (eds. V. Snasel, K. Richta, and J. Pokorny), Vol. 176, pp. 11-20, 2006.

[9] Ravindra T. and et. , "*Classification of run-length encoded binary data*", *Pattern Recognition*, Vol. 40, pp. 321-323, 2007.

[10] Rueda L. and et. , "*A Fast and Efficient Nearly-Optimal Fano Coding Scheme*", *Information Science*, Vol. 176, No. 12, pp.1656-1683, 2006.

[11] Salomon D., "*Data Compression*", (Springer) 1997.

[12] Shapira D. and et. , "*Adapting the Knuth–Morris–Pratt Algorithm for Pattern Matching in Huffman Encoded Texts*", *Information Processing and Management*, Vol. 42, pp. 429–439, 2006.

[13] Sharieh A., "*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.

[14] Vitter J., "*Dynamic Huffman Coding*", *Journal of ACM*, Vol. 15, No. 2, pp. 158-167, 1989.

[15] Witten I., "*Adaptive Text Mining: Inferring Structure from Sequences*", *Journal of Discrete Algorithms*, Vol. 2, No. 2, pp. 137-159, 2004.

[16] Witten I. and et. , "*Arithmetic Coding for Data Compression*", *Communications of the ACM, Computing Practice*, Vol. 30, No. 6, pp. 520-540, 1987.

Wiam Al Hayek was born in Amman, Jordan, on January ,1981. She received the B.S. degree from the Department of science , Al Balqa Applied University , Amman, in 2003. She received the M.S. degree from the department of computer , Amman Arab University for Graduated Studies, in 2008. I worked as computer lab supervisor during the period 2003 - 2007 within the university of Al Balqa Applied University, after that I was lecturer for different topics like (computer skills (1), computer skills (2), visual basic , Internet, web design , Programming in C++ language, Introduction to Data Base , Programming in ORACLE).