Two-Stage Algorithm for Data Compression

Prof. Nirali Thakkar
Department of Computer Engineering,
Madhuben & Bhanubhai Patel Women's Institute of Engineering
New V. V. Nagar, India

Prof. Malay Bhatt
Department of Computer Engineering, Dharamsinh Desai University
Nadiad, Gujarat, India

Abstract— **This paper proposes two stage algorithm that carries advantages of PDLZW and Arithmetic coding and compares its performance with deflate which is a well-known two-stage algorithm that combines the features of LZ77 and Huffman Coding. The PDLZW is designed by partitioning the dictionary into several dictionaries of different address spaces and sizes. With the hierarchical parallel dictionary set, the search time can be reduced significantly since these dictionaries can operate independently and thus can carry out their search operations in parallel. Arithmetic coding replaces a stream of input symbols with a single floating-point output number.**

***Keywords*——Arithmetic Coding, Lossless Data Compression, Lossy Data Compression, Parallel Dictionary LZW (PDLZW).**

I. Introduction

Data compression is a method of encoding rules that allows substantial reduction in the total number of bits to store or transmit a file [1]. It is a way to eliminate the unwanted redundancy. Data compression technique can be divided into two major families: lossless data compression and Lossy data compression.

Deflate is two-stage lossless data compression algorithm that uses the combination of LZ77 and Huffman coding. This will take advantage of both the algorithms. It is a popular compression method that was originally used in the well-known Zip and Gzip software and has since been adopted by many applications. The following figure shows the block diagram of deflate. At encoder side, the row data are compressed by LZ77 encoder. The output of LZ77 is Literals and length/distance. It is processed by Huffman Encoder which results in compressed bit stream. At decoder side, compressed data is decoded in the Huffman Decoder to construct a stream of symbols required by the LZ77 decoder. The LZ77 decoder operates reconstruct the original data.

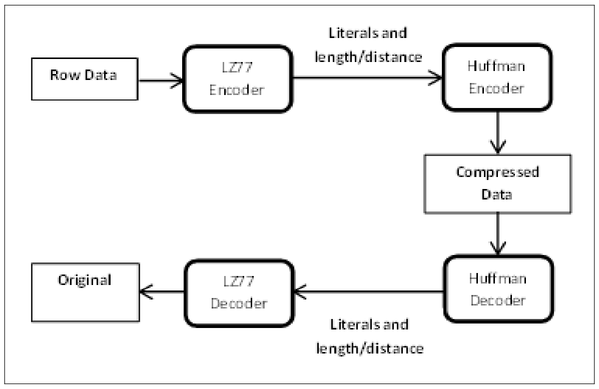

Figure 1 Block diagram of deflate

Huffman codes have to be an integral number of bits long, and this can sometimes be a problem. If the probability of a character is 1/3, for example, the optimum number of bits to code that character is around 1.6 bits. Huffman coding has to assign either one or two bits to the code and either choice lead to a longer compressed message than is theoretically possible. This non optimal coding becomes a noticeable problem when the probability of a character is very high [2]. Thus the Huffman coding always produces rounding errors while Arithmetic coding replaces a stream of input symbols with a single floating-point output number.

One of the most widely used compression methods for lossless compression is LZ77. LZ77 encoder maintains a window to the input stream and shifts the input in that window from right to left as strings of symbols are being encoded. This method is based on a sliding window. The window below is divided into two parts. The part on the left is called the search buffer. This is the current dictionary, and it always includes symbols that have recently been input and encoded. The part on the right is the look-ahead buffer, containing text yet to be encoded. An LZ77 token has three parts: offset, length, and next symbol in the look-ahead buffer. The main disadvantage of LZ77 is the size of both the buffers is very small. Increasing the sizes of the two buffers also means creating longer tokens. These will produce the higher compression ratio but it will reduce the compression efficiency [5].

LZW is a dictionary based compression, which encodes input data through establishing a string table and searching the table to identify the longest possible input data string that exists in the table. The encoded output is a sequence of the matching string's address and length. It can typically compress large English texts to about half of their original sizes. However, conventional LZW algorithm requires large amount of processing time for adjusting and searching through the dictionary [3].

The dynamic LZW (DLZW) and word-based DLZW (WDLZW) algorithms were proposed to improve searching efficiency. In DLZW, the dictionary has been initialized with different combinations of characters. It is organized in hierarchical string tables. The baseline idea is to store the most frequently used strings in the shorter table, which requires fewer bits to identify the corresponding string. The tables are updated using the move-to-front and weighting system with associated frequency counter. During the compression time, after the longest matching string is recognized in the table, it is moved to the first position of its block. The table updating process is based on the least recently used (LRU) policy to ensure that frequently used strings are kept in the smaller tables. This is to minimize the average number of bits required to code a string when compare with a single table implementation [3].

The WDLZW algorithm is a modified version of DLZW that focuses on text compression by identifying each word in the text and make it a basic unit (symbol). The algorithm encodes the input word into literal codes and copy codes. If the search for a word has failed, it is sent out as a literal code, which is its original ASCII code preceded by other codes for identification. The copy code is the address of the matching string in the string table. However, both algorithms are too complicated. To improve this, parallel dictionary LZW (PDLZW) was proposed. Since not all entries of the DLZW dictionary contains the same word size, this leads to the need of the entire dictionary search for every character. Consequently, the PDLZW has designed to overcome this problem by partitioning the dictionary into several dictionaries of different address spaces and sizes. With the hierarchical parallel dictionary set, the search time can be reduced significantly since these dictionaries can operate independently and thus can carry out their search operation in parallel [3].

II. PRAPOSED APPROCH

In this section, a new two-stage algorithm is proposed. In this approach, the row data is given to the PDLZW encoding algorithm. The output of the PDLZW is given to the Arithmetic coding for further compression. The decompression process is totally reverse. Figure 2 shows the block diagram of this new two-stage algorithm.

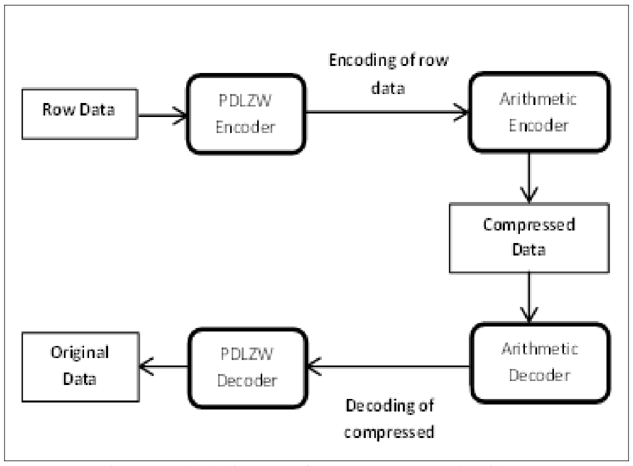

Figure 2 Block diagram of a new two-stage algorithm

A. *PDLZW Encoding Algorithm*

As shown in figure 2, the row data is given to PDLZW encoding algorithm. PDLZW algorithm is a LZW based implementation using a parallel dictionary set. It partitions one large dictionary into several small variable-word-width dictionaries. Searching in parallel through small dictionaries would require less amount of processing time than searching sequentially through one large-address-space dictionary.

The PDLZW encoding algorithm is based on a parallel dictionary set that consists m of small variable-word-width dictionaries, numbered from 0 to m-1, each of which increases its word width by one byte. More precisely, dictionary 0 has one byte word width, dictionary 1 two bytes, and so on. The following show the detailed operation of the PDLZW encoding algorithm. PDLZW dictionary initialized with the input symbols. ∑ represents the set of input symbols and |∑| indicate the number of input symbols. The PDLZW compression and decompression algorithms are shown in [1] and [4].

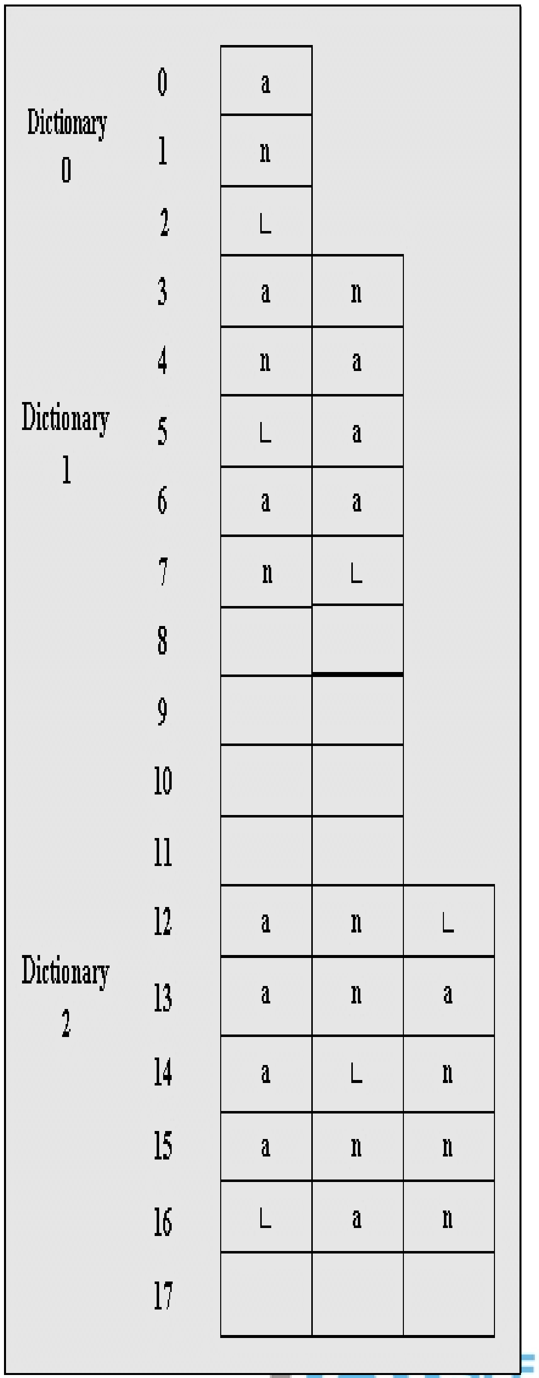

Figure 3 Dictionary Structure of PDLZW Algorithm

The table I shows the example of the PDLZW Encoding Algorithm. Assume that the alphabet set ∑ is {a, n, ∟} and the input file contains anan∟aanann∟an. The number of input symbols is 3. The dictionary set initially contains only all single character: a, n and ∟. Here for convince we will use '∟' to indicate space.

TABLE I
ENCODING PROCESS OF PDLZW ALGORITHEMS

Input	In dictionary?	New Entry	Output
a	T		
an	F	an – 3	a – 0
n	T		
na	F	na – 4	n – 1
a	T		
an	T		
an∟	F	an∟- 12	an – 3
∟	T		
∟a	F	∟a – 5	∟ – 2
a	T		
aa	F	aa – 6	a – 0
a	T		
an	T		
ana	F	ana – 13	an – 3
a	T		
an	T		
ann	F	ann – 14	an – 3
n	T		
n∟	F	n∟ – 7	n – 1
∟	T		
∟a	T		
∟an	F	∟an – 8	∟a – 5
n	T		n – 1

B. Arithmetic Encoding

Arithmetic coding bypasses the idea of replacing an input symbol with a specific code. More bits are needed in the output number for longer, complex messages.

The output of the PDLZW encoding algorithm is {0, 1, 3, 2, 0, 3, 3, 1, 5, 1} and the corresponding ASCII characters of these values are {NULL, SOH, ETX, STX, NULL, ETX, ETX, SOH, ENQ, SOH}. As shown in figure 2, the output of the PDLZW is given to the Arithmetic encoder as an input. The output from an arithmetic coding process is a single number less than 1 and greater than or equal to 0. This single number can be uniquely decoded to create the exact stream of symbols that went into its construction. To construct the output number, the symbols are assigned set probabilities.

Now next step is to find out the probability and range of each symbol. Table 2 shows the probability and range calculation of each symbol. The table also shows the ASCII value of each character. These values are used by the arithmetic encoding algorithm to calculate the low and high values of the symbols. The characters shown in table are non-printable characters. So it can be represented by some special name.

TABLE III
PROBABLITY AND RANGE OF THE

ASCII Value	Character	Probability	Range
0	NULL	2/10(= 0.20)	0.00 ≥ r > 0.20
1	SOH	3/10(= 0.30)	0.20 ≥ r > 0.50
2	STX	1/10(= 0.10)	0.50 ≥ r > 0.60
3	ETX	3/10(= 0.30)	0.60 ≥ r > 0.90
5	ENQ	1/10(= 0.10)	0.90 ≥ r > 1.00

The following table shows the encoding process of arithmetic coding. Initially, the low value is 0.0 and high value is 1.0. The range is difference between high and low values. Arithmetic encoding algorithm is shown in [2]. Next step is to calculate the low and high values as per the algorithm.

TABLE IIIII
ENCODING PROCESS OF ARITHMETIC ENCODING

Character	range	Low	High
		0.0	1.0
NULL	1.00	0.0	0.2
SOH	0.20	0.04	0.1
ETX	0.06	0.076	0.094
STX	0.018	0.085	0.0868
NULL	0.0018	0.085	0.08536
ETX	0.00036	0.085216	0.085324
ETX	0.000108	0.0852808	0.0853132
SOH	0.0000324	0.08528728	0.085297
ENQ	0.00000972	0.085296028	0.085297
SOH	0.000000972	0.085296222	0.085295514

Now next step is to calculate the tag value of last symbol. So the final low value is 0.085296222 and high value is 0.085295514. Now find out the tag value for this example. Tag is the midpoint of the given interval. It forms a unique representation for the sequence.

T = (high + low) ÷ 2
T = (0.085295514 + 0.085296222) ÷ 2
T = 0.085295681

Figure 4 Calculation of Tag value.

This tag value is converted into the bits using bitio functions [2]. This will create the bit file.

C. Arithmetic Decoding Algorithm

The following table shows the decoding process with example. Since 0.085295681 falls between 0.0 and 0.2, the first character must be NULL. The decoding algorithm is shown in [2].

TABLE IVV
DECODING PROCESS OF ARITHMETIC ENCODING

Encoded Number	Output Symbol	Low	High	Range
0.085295681	NULL	0.0	0.2	0.2
0.426481112	SOH	0.2	0.5	0.3
0.75493704	ETX	0.6	0.9	0.3
0.5164568	STX	0.5	0.6	0.1
0.164568	NULL	0.0	0.2	0.2
0.82284	ETX	0.6	0.9	0.3
0.7428	ETX	0.6	0.9	0.3
0.476	SOH	0.2	0.5	0.3
0.92	ENQ	0.9	1.0	0.1
0.2	SOH	0.2	0.5	0.3
0.0				

D. PDLZW Decoding Algorithm

As shown in figure 2, the output of the Arithmetic decoder is given to as an input of PDLZW decoder. The operation of the PDLZW decoding algorithm can be illustrated by the following example. Assume that the alphabet set ∑ is {a, n, ∟} and input compressed codewords are {0, 1, 3, 2, 0, 3, 3, 1, 5, 1}. Initially, the dictionaries numbered from 1 to 3 shown in Figure 3 are empty. By applying the entire input compressed codewords to the algorithm, it will generate the same content as is shown in Fig. 1 and output the decompressed substring {a, n, an, ∟, a, an, an, n, ∟a, n}. It the advantage of PDLZW algorithm is that there is no need to pass the whole dictionary to the PDLZW decoder. The dictionary can be built exactly as it was during the PDLZW encoder using input stream as data. This will increase the efficiency of the algorithm.

After compression, the next step is to decompress the compressed data to get original one. As shown in figure 2, first arithmetic decoder will decompress the floating point number and generate the integer numbers. At decoder side, first bits are converted to floating point value. The next step is to convert this floating point value in symbols.

TABLE VI
DECODING PROCESS OF PDLZW ALGORITHEMS

Old_Code	New_Code	Output	Character	Dictionary
	a	a	a	
a	n	n	N	an – 3
n	3	an	A	na – 4
3	∟	∟	∟	an∟ – 12
∟	a	a	A	∟a – 5
a	3	an	A	aa – 6
3	3	an	A	ana – 13
3	n	n	N	ann – 14
n	5	∟a	∟	n∟ – 7
5	n	n	n	∟an – 15

III. Conclusions

The two-stage compression algorithm combines the features of PDLZW algorithm and arithmetic coding. PDLZW is the better then the other dictionary based algorithm(LZ77 in deflate) in terms of dictionary structure and arithmetic coding produces the better results compare to the Huffman coding which is used in deflate. So the combination of both the algorithms will produce the higher compression-ratio compares to deflate.

Acknowledgment

The author would like to thank the anonymous reviewers and the communicating editor for their useful comments which improved the presentation of paper.

References

[1] M. B. Lin, Jang-Feng Lee and Gene Eu Jan, “A Lossless Data Compression and Decompression Algorithm and Its Hardware Architecture”, *IEEE Transactions on VLSI Systems*, vol. 14, No. 9, pp. 925-936, Sep. 2006.

[2] M. Nelson and Jean-Loup Gailly , "*The Data Compression Book",* 2nd ed., BPB publications, 1996.

[3] P. Vichitkraivin and O. Chitsobhuk, "An Improvement of PDLZW Implementation with a Modified WSC Updating Technique on FPGA", *World Academy of Science, Engineering and Technology*, 2009.

[4] M.B. Lin, "A Hardware Architecture for the LZW Compression and Decompression Algorithms Based on Parallel Dictionary," *Journal of VLSI Signal Processing 26*, pp. 369-381, 2000.

[5] D. Salomon, "*Data Compression the Complete Reference*", 4th ed., Springer, 2007.