A Probabilistic Compression Algorithm based on Symbols Position

Carlos Rincón*, Gerardo Pirela-Morillo*, David Bracho*, Alfredo Acurero* and Jesús Carretero†
*Computer Science Department, Universidad del Zulia, Maracaibo, Venezuela
Email: see http://www.fec.luz.edu.ve/dep_computacion.htm †Computer Architecture, Communications and Systems Group, Universidad Carlos III de Madrid, Madrid, España
Email: see: http://arcos.inf.uc3m.es/˜jcarrete/

***Abstract*—The purpose of this research was to design a new lossless compression algorithm based on Shannon's 1948 concepts and fundamentals of information theory. After reviewing these principles and the most commonly used lossless compression algorithms, a new compression algorithm was designed using symbols position to generate compressed codes. As a result of the study, it was possible to prove the generation of optimal codes by the new algorithm, putting forth future works to describe the algorithm's performance.**

I. Introduction

Information is being generated ever since human beings developed the ability to analyze what goes on in their surroundings. Many branches of science give a definition of "information", but it wasn't until mid XIX century, as a consequence of the emergence of automated computing systems, that mechanisms were put forth to quantify information.

Despite computing systems fundamental task of generating information, most computer systems students and professionals define information as a meaningful, structured set of data and only know of the bit as the minimal unit for storing and transmitting information.

In 1948, Claude Shannon [1] introduced a mathematical method to quantify information generated by a data source. Shannon is considered to have founded what is now known as Information Theory with such a method, which is based upon a relatively simple concept. Considering that every event occurs with some probability, the quantity of information generated by such an event can be measured by an inverse function of its occurrence probability. Hence, a low-probability event occurring generates more information than a high-probability event.

Mankind's need to process ever-increasing amounts of data has generated problems of overwhelming sums of information, which require to be stored and transmitted. This phenomenon has driven an ever-increasing development over time of computational storage and transmission capabilities to support informatics systems. However, the amount of information generated exceeds current computational systems capabilities, which in turn drives the need for new technology to effectively take advantage of computational resources.

To solve this problem, computer scientists devised methods based on techniques used in the past to solve similar problems. Through efficient coding of a message's symbols, a representation of the original message can be obtained using less information units.

Data compression is the process that allows optimal representation of data generated by a source, minimizing the quantity of information that the source yields. According to Sayood [2], compression algorithms can be classified as loss algorithms (if the resulting representation is not exactly the same as the original message) and lossless algorithms (if the resulting representation is exactly the same as the original message.) The use of one set of algorithms over the other depends on the characteristics of the information to be shared.

In the early 50's, David Huffman [3], student of Robert Fano (Shannon's colleague,) introduced a technique that allows generating optimal compression codes. This lossless technique is known as Huffman's Compression Algorithm and is still in use despite the introduction of many other optimal code generating algorithms.

Huffman designed and introduced his algorithm in 1952 based on information theory concepts put forth by his professor (Claude Shannon) at MIT in 1942. Despite

all the time that has passed and all the newer algorithms that have been developed since, Abraham Bookstein [4] concluded in his paper "Is Huffman Coding Dead?" that Huffman's algorithm can be used currently to solve different problems in informatics which may require data compression.

One such area where Huffman's algorithm is currently used is in academia, as evidenced by the works of Mohamed Hamada [5] and James Keeler [6], where Huffman's algorithm was implemented to demonstrate concepts implicit in information theory.

Others works like [7], [8], [9], [10], [11], [12], [13], [14], [15], [16], [17], [18] and others, allows to conclude that lossless compression algorithms are used today in many fields of computer science, making the design of a new method to generate optimal codes based on lossless compression technics an important achievement.

This current work consisted in developing a compression algorithm based on information theory to provide the computer science community with yet another tool to implement data compression solutions which minimizes the amount of bits, needed to represent information.

II. Algorithm Design

The probabilistic algorithm we propose was designed using the symbols position in the message to generate codes, thus avoiding having to re-write the entire message with different codes.

Like every other probabilistic algorithm, ours generates optimal codes using symbols frequency of occurrence in the message as parameter. Shannon's concept of information allows assigning smaller codes to higher frequency symbols, resulting in a minimization of bit size to represent the original message, yielding certain degree of compression.

A. Compression Process

The algorithm uses symbols position in the message instead of generating a new substitution code. To that end, a bit word is used to store the position of a particular symbol in the message. Each symbol is associated with such a binary string and it is this information, which is ultimately stored.

Figure 1 depicts this compression process. The corresponding algorithm has seven steps, as follows. 1) Compute symbols occurrence frequency in the message; 2) create a priority queue of symbols keyed descendingly by frequency; 3) extract highest-frequency symbol from queue; 4) add to compressed messages bit string an n-bit word with n equal to the original message size, 1 on the position of the message where the extracted symbol occurs, and 0 where it does not; 5) delete current symbol from original message; 6) repeat steps 3 to 5 until queue is empty; 7) the concatenation of all symbols associated bit words is the compressed representation of the original message.

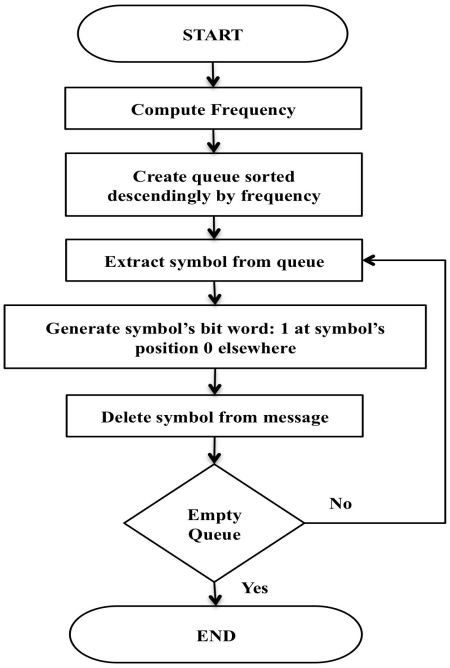

Fig. 1. Compression Process

B. Decompression Process

To reconstruct the original message all we need are the compressed message and the size in bits used to represent each symbol in the original message. Since the binary string that represents the compressed message was built in descending order of occurrence frequency, the decompression process is carried out inversely, starting from the least frequent symbol, taking into account the number of bits it occupies in the binary string (represented by the decimal equivalence of its bit-word) to systematically delete them from compressed message as the reconstruction process advances. At the same time, a string of characters is built with the symbols associated to the content of each bit string segment. Figure 2 depicts this process.

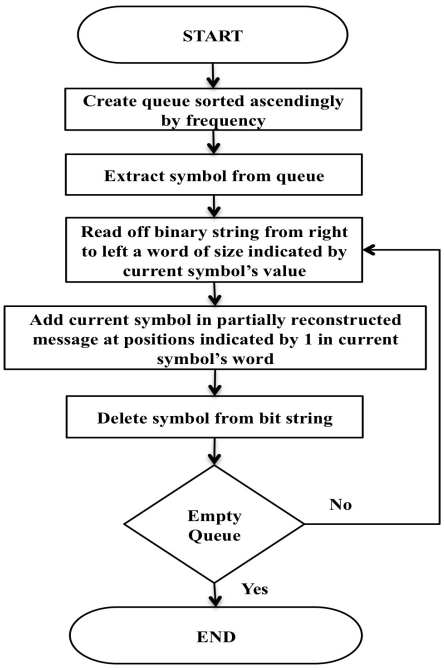

Fig. 2. Decompression Process

III. Mathematical Analysis of Algorithm Performance

To analyze the efficiency of the method we proposed, we need to establish some assumptions as to the diversity and frequency of the distinct characters that occur within the original message.

Let us name the original message w (of character length n) and the corresponding compressed message w^c. Furthermore, lets say that there occur m distinct characters in w. The ratio between the length of the compressed message and the bit-length of the original message can be used as a measure of compression efficiency. Since the compressed message is a binary string and assuming the original message is coded in ASCII, such ratio becomes:

$$\frac{|w^c|}{8 * n}$$

Now, assuming equal frequency of occurrence of all m distinct characters along w, we can compute the length of w^c according to the following sum:

$$|w^c| = n + \left(n - \frac{n}{m}\right) + \left(n - \frac{2 * n}{m}\right) + ... + \frac{n}{m} = \sum_{i=0}^{m-1} \left(n - \frac{i * n}{m}\right) = \frac{n * (m+1)}{2}$$

The efficiency ratio becomes:

$$\frac{(m+1)}{16}$$

On one hand, this ratio indicates that for m=1 (the original message is just a repetition of a single character n times), the efficiency ratio is 1/8 (which translates to an 87.5% storage reduction). On the other hand, for m=n (the original message is made of n distinct characters, all with frequency 1), the compressed message is actually far longer than the original message ($|w^c| \in \Theta(n^2)$). Moreover, for m $\geq$ $15, |w^c| \geq 8|w|$ (the compressed message is longer than the bit representation of the original message).

The cases described above are in fact the best and worst case scenarios (respectively) for our algorithm's performance. However, a uniform distribution of the m distinct characters along w is a rather unreasonable assumption, especially for intelligible messages. Perhaps a more appropriate assumption might be a Multinomial distribution for human-readable text or a Poisson distribution for more general text; nevertheless, some work needs to be done to tune the corresponding parameters of the assumed distributions in either case. Hence, further work is indeed required to better assess the average case performance scenario for the algorithm we present.

IV. An example of the Implementation of the Algorithm

The following example illustrates the compression process described above.

Symbol size: 8 bits.

Message: ABCABCACBAAABCCCCBDF
Message size = 20

1) Frequency: A(6), B(5), C(7), D(1), F(1)

2) Sort by frequency: C(7), A(6), B(5), D(1), F(1)

3) Extract from queue: Symbol C

4) 20-bit word for symbol C:

A	B	C	A	B	C	A	C	B	A	A	A	B	C	C	C	C	B	D	F
0	0	1	0	0	1	0	1	0	0	0	0	0	1	1	1	1	0	0	0

5) Delete symbol C from message:
New message: ABABABAAABBDF
Message size = 13

6) Queue not empty, go to step 3.

3) Extract from queue: Symbol A

4) 13-bit word for symbol A:

A	B	A	B	A	B	A	A	A	B	B	D	F
1	0	1	0	1	0	1	1	1	0	0	0	0

5) Delete symbol A from message:
New message: BBBBBDF
Message size = 7

6) Queue not empty: go to step 3.

3) Extract from queue: Symbol B

4) 7-bit word for symbol B:

B	B	B	B	B	D	F
1	1	1	1	1	0	0

5) Delete symbol B from message:
New message: DF
Message size = 2

6) Queue not empty: go to step 3.

3) Extract from queue: Symbol D

4) 2-bit word for symbol D:

D	F
1	0

5) Delete symbol D from message:
New message: F
Message size = 1

6) Queue not empty: go to step 3.

3) Extract from queue: Symbol F

4) 1-bit word for symbol F:

F
1

5) Delete symbol F from message:
New message: empty
Message size = 0

6) Queue empty: go on to step 7.

7) Concatenate the bit words generated:
00100101000001111000101010111000011111100101
Compressed message size = 43 bits.

The resulting compressed representation of the original message is the concatenation of all the symbols associated bit words in the order in which they were generated:

C= 20; A=13; B=7; D=2; F=1

00100101000001111000	1010101110000	1111100	10	1
C	A	B	D	F

The size of the compressed message is the sum of the sizes of each symbols bit word (measured in bits), while the size of the original message using a standard ASCII representation (8 bits per character) is 8*size of message. For the example described above, the size of the original message is 8*20 = 160 bits, while the size of the compressed message is 20+13+7+2+1 = 43 bits, which is equivalent to roughly 27% of the original message size (a 73% storage reduction).

Given symbols order and decimal representation of bit word during compression process:

C= 20; A=13; B=7; D=2; F=1;

and the compressed message:

00100101000001111000101010111000011111100101

To retrieve the original message from the compressed message, we proceed as follow:

1) Create queue of symbols in reverse order.
F = 1; D = 2; B = 7; A = 13; C = 20;

2) Extract symbol from queue: Symbol F.

3) Read off binary string from right to left a word of size indicated by current symbols value:
Symbol F's size = 1
Symbol F's word = 1

4) Add current symbol in partially reconstructed

message at positions indicated by 1 in current symbols word:
Message: F

5) Delete current symbol's word from binary string:
00100101000001111000101010111000011111 0010

6) Queue not empty: go to step 2.

2) Extract symbol from queue: Symbol D.

3) Read off binary string from right to left a word of size indicated by current symbols value:
Symbol D's size = 2
Symbol D's word = 10

4) Add current symbol in partially reconstructed message at positions indicated by 1 in current symbols word:
Message: DF

5) Delete current symbols word from binary string:
00100101000001111000101010111000011111 00
6) Queue not empty: go to step 2

2) Extract symbol from queue: Symbol B.

3) Read off binary string from right to left a word of size indicated by current symbols value:
Symbol B's size = 7
Symbol B's word = 1111100

4) Add current symbol in partially reconstructed message at positions indicated by 1 in current symbols word:
Message: BBBBBDF

5) Delete current symbols word from binary string:
00100101000001111000101010111 0000

6) Queue not empty: go to step 2

2) Extract symbol from queue: Symbol A.

3) Read off binary string from right to left a word of size indicated by current symbols value:
Symbol A's size = 13
Symbol A's word = 1010101110000

4) Add current symbol in partially reconstructed message at positions indicated by 1 in current symbols word:
Message: ABABABAAABBDF

5) Delete current symbols word from binary string:
00100101000001111000

6) Queue not empty: go to step 2

2) Extract symbol from queue: Symbol C.

3) Read off binary string from right to left a word of size indicated by current symbols value:
Symbol C's size = 20
Symbol C's word = 00100101000001111000

4) Add current symbol in partially reconstructed message at positions indicated by 1 in current symbols word:
Message: ABCABCACBAAABCCCCBDF

5) Delete current symbols word from binary string:
empty

6) Queue empty: stop.

The reconstruction process ends successfully, retrieving the original message from the compressed message, starting from the last symbol of the generated alphabet all the way back to the first, without any information loss.

V. Conclusion and Future Work

We described a new probabilistic, lossless compression algorithm. Its innovation derives from using symbols positions in the original message to generate symbols codes instead of quantity of information a symbol contributes to the message, which most other similar algorithms use. Utilizing original message size and symbols positions as key parameters guarantees optimal code generation.

We propose that a preliminary analysis be done of the effect that parameters inherent to information theory (message size, entropy, etc.) have on the algorithm. Further analysis needs to include benchmarking the algorithms performance against other commonly used probabilistic, lossless compression algorithms like Huffman, Shannon-Fano, and others.

References

[1] C. E. Shannon, "A mathematical theory of communication," *Bell system technical journal*, vol. 27, 1948.

[2] K. Sayood, *Introduction to data compression*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1996.

[3] D. A. Huffman, "A method for the construction of minimum-redundancy codes," *Proceedings of the Institute of Radio Engineers*, vol. 40, no. 9, pp. 1098–1101, September 1952.

[4] A. Bookstein, S. T. Klein, and T. Raita, "Is huffman coding dead? (extended abstract)," in *SIGIR '93: Proceedings of the 16th annual international ACM SIGIR conference on Research and development in information retrieval*. New York, NY, USA: ACM, 1993, pp. 80–87.

[5] M. Hamada, "Web-based tools for active learning in information theory," in *SIGCSE '07: Proceedings of the 38th SIGCSE technical symposium on Computer science education*. New York, NY, USA: ACM, 2007, pp. 60–64.

[6] J. Keeler, "Graphical implementation of huffman and arithmetic coders," *J. Comput. Small Coll.*, vol. 19, no. 5, pp. 289–290, 2004.

[7] Y.-H. Chan, "A lossless coding scheme for encoding color-indexed video sequences," *Neural Networks and Signal Processing, 2008 International Conference on*, pp. 676–681, June 2008.

[8] W. Huang, W. Wang, and H. Xu, "A lossless data compression algorithm for real-time database," *Intelligent Control and Automation, 2006. WCICA 2006. The Sixth World Congress on*, vol. 2, pp. 6645–6648, 0-0 2006.

[9] L. Yun, Q. Chen-yi, and W. Xun, "A new lossless compression algorithm for vector maps," *Computer Science and Computational Technology, 2008. ISCSCT '08. International Symposium on*, vol. 1, pp. 347–351, Dec. 2008.

[10] J. Zhang and G. Liu, "A novel lossless compression for hyperspectral images by adaptive classified arithmetic coding in wavelet domain," *Image Processing, 2006 IEEE International Conference on*, pp. 2269–2272, Oct. 2006.

[11] F. Marcelloni and M. Vecchio, "A simple algorithm for data compression in wireless sensor networks," *Communications Letters, IEEE*, vol. 12, no. 6, pp. 411–413, June 2008.

[12] Y. Shi, L. He, Y. Wen, B. Li, and Z. Chen, "An efficient lossless compression algorithm for well logging result drawings," *Image and Graphics, 2007. ICIG 2007. Fourth International Conference on*, pp. 200–204, Aug. 2007.

[13] J. Zhang and G. Liu, "An efficient reordering prediction-based lossless compression algorithm for hyperspectral images," *Geoscience and Remote Sensing Letters, IEEE*, vol. 4, no. 2, pp. 283–287, April 2007.

[14] A. Kattan and R. Poli, "Evolutionary lossless compression with gp-zip," *Evolutionary Computation, 2008. CEC 2008. (IEEE World Congress on Computational Intelligence). IEEE Congress on*, pp. 2468–2472, June 2008.

[15] A. Gu and A. Zakhor, "Lossless compression algorithms for hierarchical ic layout," *Semiconductor Manufacturing, IEEE Transactions on*, vol. 21, no. 2, pp. 285–296, May 2008.

[16] L. Bai, M. He, and Y. Dai, "Lossless compression of hyperspectral images based on 3d context prediction," *Industrial Electronics and Applications, 2008. ICIEA 2008. 3rd IEEE Conference on*, pp. 1845–1848, June 2008.

[17] A. Gu and A. Zakhor, "Lossless compression algorithms for post-opc ic layout," *Image Processing, 2007. ICIP 2007. IEEE International Conference on*, vol. 2, pp. II –357–II –360, 16 2007-Oct. 19 2007.

[18] V. Dai and A. Zakhor, "Lossless compression of vlsi layout image data," *Image Processing, IEEE Transactions on*, vol. 15, no. 9, pp. 2522–2530, Sept. 2006.