Journal of Kerbala University , Vol. 6 No.2 Scientific. 2008

Role of Run Length Encoding on Increasing Huffman Effect in Text Compression

B.A. Al-hmeary
م.م. بيادر عباس الحميري
جامعة بابل / كلية العلوم

Abstract:

Most digital data are not stored in the most compact form. Rather, they are stored in whatever way makes them easiest to use, such as: ASCII text from word processors, binary code that can be executed on a computer, individual samples from a data acquisition system, etc. Typically, these easy-to-use encoding methods require data files about twice as large as actually needed to represent the information. Data compression is the general term for the various algorithms and programs developed to address this problem. A compression program is used to convert data from an easy-to-use format to one optimized for compactness. Likewise, an decompression program returns the information to its original form.

This research aims to appear the effect of a simple lossless compression method , RLE or Run Length Encoding , on another lossless compression algorithm which is the Huffman algorithm that generates an optimal prefix codes generated from a set of probabilities. While RLE simply replaces repeated bytes with a short description of which byte to repeat it.

1.1 Introduction [10,4,7,2]:

Finding the optimal way to compress data with respect to resource constraints remains one of the most challenging problems in the field of source coding. Data compression (or source coding) is the process of creating binary representations of data which require less storage space than the original data.

The key factors in reducing the amount of data storage is:

1- Getting rid of redundant data. This involves determining the parts of the data that are not required.

2- Identifying irrelevant data. This involves identifying the parts of the data which are perceived to be irrelevant.

3- Converting the data into a different format. This will typically involve changing the way that the data is processed and stored.

4- Reducing the quality of the data. Often the user does not require the specified quality of the data.

In order to discuss the relative merits of data compression techniques, a framework for comparison must be established. There are two dimensions along which each of the schemes discussed here may be measured, algorithm complexity and amount of compression. When data compression is used in a data transmission application, the goal is speed. Speed of transmission depends upon the number of bits sent, the time required for the encoder to generate the coded message, and the time required for the decoder to recover the original ensemble. In a data storage application, although the degree of compression is the primary concern, it is nonetheless necessary that the algorithm be efficient in order for the scheme to be practical.

Several common measures of compression have been suggested: redundancy [Shannon and Weaver 1949], average message length [Huffman 1952], and compression ratio [Rubin 1976;

Ruth and Kreutzer 1972]. Related to each of these measures are assumptions about the characteristics of the source. It is generally assumed in information theory that all statistical parameters of a message source are known with perfect accuracy [Gilbert 1971]. The most common model is that of a discrete memoryless source; a source whose output is a sequence of letters (or messages), each letter being a selection from some fixed alphabet a,... The letters are taken to be random, statistically independent selections from the alphabet, the selection being made according to some fixed probability assignment *p*(a),... [Gallager 1968].

There is another categorization of data compression schemes with respect to message and codeword lengths, these methods are classified as either static or dynamic. A *static* method is one in which the mapping from the set of messages to the set of code words is fixed before transmission begins, so that a given message is represented by the same codeword every time it appears in the message ensemble. The classic static defined-word scheme is Huffman coding [Huffman 1952]. In Huffman coding, the assignment of code words to source messages is based on the probabilities with which the source messages appear in the message ensemble. Messages which appear more frequently are represented by short code words; messages with smaller probabilities map to longer code words. These probabilities are determined before transmission begins.

A code is *dynamic* if the mapping from the set of messages to the set of code words changes over time, for example, dynamic Huffman coding.

2.1 Huffman Method [3,9,6,8]

David Huffman developed the Huffman algorithm as a student in a class of information theory at MIT in 1950. The Huffman algorithm is very simple and is most easily described in terms of how it generates the prefix-code tree.

The algorithm of Huffman is:-

1.Start with as many leaves as there are symbols.

2.Push all leaf nodes into the heap.

3.While there is more than one node in the heap:

-Remove two nodes with the lowest weight from the heap.

-Put the two nodes into the tree, noting their location.

-If parent links are used, set the children of any internal nodes to point at their parents.

-Create a new internal node, using the two nodes as children and their combined weight as the weight.

-Push the new node into the heap.

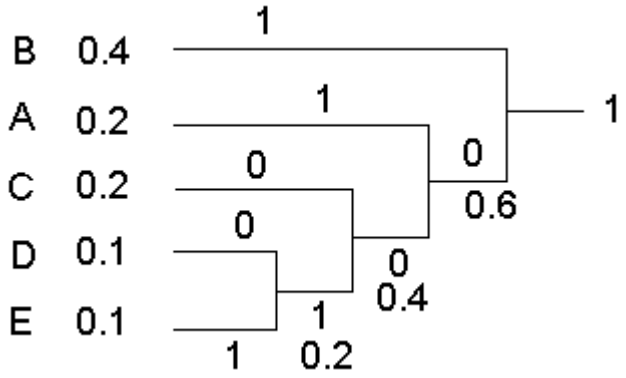
4.The remaining node is the root node.

2.2 Main Properties[5] :-

The frequencies used can be generic ones for the application domain that are based on average experience, or they can be the actual frequencies found in the text being compressed. (This variation requires that a frequency table or other hint as to the encoding must be stored with the compressed text; implementations employ various tricks to store these tables efficiently).

Huffman coding is optimal when the probability of each input symbol is a negative power of two. Prefix-free codes tend to have slight inefficiency on small alphabets, where probabilities often fall between these optimal points. Expanding the alphabet size by coalescing multiple symbols into "words" before Huffman coding can help a bit. The worst case for Huffman coding can happen when the probability of a symbol exceeds 2^{-1} making the upper limit of inefficiency unbounded.

For an example on the Huffman coding, consider the following message probabilities, and the Huffman tree in figure(1)



Figure(1) : Huffman Tree

The following table illustrates the codes .

Table (1) Huffman Coding

Symbol	Probability	Code 1
A	0.2	01
B	0.4	1
C	0.2	000
D	0.1	0010
E	0.1	0011

3.1 Run-Length Encoding:-

Data files frequently contain the same character repeated many times in a row. For example, text files use multiple spaces to separate sentences, indent paragraphs, format tables & charts, etc.

Run-length encoding is a simple method of compressing these types of files [1,9,6,3].

Figure (2) illustrates run-length encoding for a data sequence having frequent runs of zeros. Each time a zero is encountered in the input data, two values are written to the output file. The first of these values is a zero, a flag to indicate that run-length compression is beginning. The second value is the number of zeros in the run. If the average run-length is longer than two, compression will take place. On the other hand, many single zeros in the data can make the encoded file larger than the original. Many different run-length schemes have been developed. For example, the input data can be treated as individual bytes, or groups of bytes that represent something more elaborate, such as floating point numbers. Run-length encoding can be used on only one of the characters (as with the zero below), several of the characters, or all of the characters.

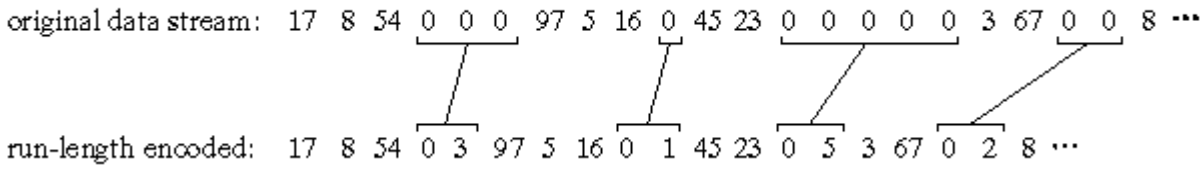
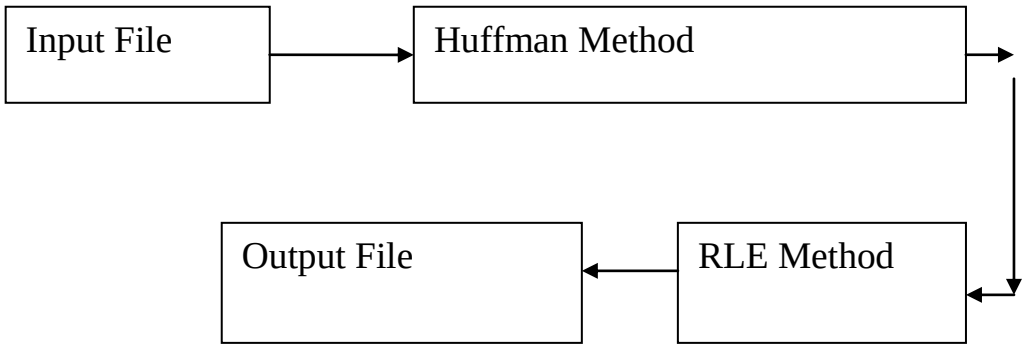


Figure (2) Run Length Encoding Example

Example of run-length encoding as shown in figure(2): Each run of zeros is replaced by two characters in the compressed file: a zero to indicate that compression is occurring, followed by the number of zeros in the run.

4.1 Proposed compression method:-

Figure (3) illustrates the block diagram of the proposed compression method.



Figure(3) Block Diagram of Proposed Method

From the above figure, we can see that the proposed compression method works as the following:-

1- Taking the input file, that is a random sequence of English alphabet symbols, computing the probability for each symbol.

2- Applying the Huffman coding of the sequence of probabilities, the result is a string of 0 and 1 bits.

3- Applying the RLE method on the string of 0 and 1 bits(the result of Huffman method). the RLE is applied after dividing the string of 0 and 1 into 8-block each and transmitting each into a byte. The RLE is applied on the bytes (0 or 255), which 0 came from a sequence of eight zeros and 255 came from a sequence of eight ones. Here the RLE is applied only the 0 and 255 bytes not on all the bytes in the string.

4- The final compressed file contains:-

a- the number of symbols.

b- the symbols.

c- the code words of each symbol.

d- the final strings resulted from applying the RLE method on the final code after substituting the codeword of each symbol in the input file.

5.1 Decompression of the proposed compression method:-

The number of symbols, each symbol, and the code words for each is distinguished. The rest of the compressed file contains on the RLE strings. After applying the decompression of this method, we get the final code that is read bit by bit in a sequential order and is compared with the code words to get the original text.

In the decompression of the RLE, only the bytes that are followed by a 0 or 255 byte is concerned as the number that represents the repeation of that byte.

6.1 Results:-

The following table illustrates the test sets of data which is used on the proposed compression method:-

In this research the test sets of data files, as shown in table (2), includes a sequence of English alphabet letters (random sequence).The compression ratio is calculated by

$$(100 - \frac{\text{(File size)}}{\text{(File size (2))}})*100\%$$

Where, File size represents the original file size including the random sequence of English alphabet letters, while File size (2) represents the results after applying the proposed compression method where the RLE method was applied.

Table (2) Test sets of data

ile	File size	File size (1)	File size (2)	Compression Ratio
File1	81 byte	25 byte	21 byte	72.84%
File2	100 byte	36 byte	36 byte	63.00%
File3	884 byte	182 byte	87 byte	90.04%
File4	210 byte	47 byte	32 byte	84.28%
File5	99 byte	34 byte	27 byte	71.17%
File6	204 byte	49 byte	36 byte	81.86%
File7	1.2 KB	226 byte	86 byte	92.42%
File8	97 byte	27 byte	19 byte	79.38%
File9	250 byte	56 byte	33 byte	86.40%
File10	296 byte	71 byte	57 byte	80.40%
File11	292 byte	70 byte	54 byte	79.39%
File12	427 byte	254 byte	208 byte	51.05%
File13	300 byte	140 byte	140 byte	53.00%

The File size (1) represents the results on the data after applying the Huffman algorithm.
The decompression process on the compressed data files in both methods, applying Huffman algorithm or applying the proposed method, returns the original uncompressed data files.

6.2 Discussion & Conclusion:-

1- The proposed compression method is lossless here both of the Huffman and RLE methods are lossless which is useful in text compression since losing a single character can in the worst case make the text dangerously misleading. This means increasing compression ratio without losing information.

2- When the compressed file contains on a longer sequence of frequented symbols, it has a high effect on the compression ratio as the proposed compression method gives better results.

3- As illustrated in table (2), it can be concluded that, when the RLE method is applied with the Huffman algorithm, if it does not decreases the file size it will not increases it. Which indicates that the RLE method has an High effect on the Huffman method when applied together.

References:

1- A. W. Berger and others," **A Hybrid Coding Strategy for Optimized Test Data Compression",**University of Innsbruck, Austria, Proceedings IEEE International Test Conference, Charlotte, NC, USA, September 30 – October 2, 2003.
2- D. Lelewer and others," **Data Compression**", ACM Press Newyork,NY,USA,1987.
3- D. Salomon , " **Data Compression the Complete Reference**" spring verlag Newyork, USA, 1998.
4- G. Kempe, "**Computer Science Honours Research Report Compression and Computational Gene Finding**", 1 November , 2002.
5- From Wikipedia, the free encyclopedia," **Huffman coding**", GNU Free Documentation license, January 19,1996.
6- J. M. Pullen ," **Data Compression, Security Principles Integrity, Appropriate Use** ",2/3/03 © 2003.
7- R. He **,"Indexing Compressed Text",** A thesis, Waterloo, Ontario, Canada, 2003 .
8- R. Müller , **"Image Compression"** Part II: Image Processing Computer Graphics and Image Processing , Winter Semester 2003/04.
9- S. W. Smith, A sample chapter from: **The Scientist and Engineer's Guide to Digital Signal Processing",** 1997.
10- W. KEONG NG," **Lossless and Lossy Data Compression"** Nanyang Technological University, Singapore, 1996.