Modified Run Length Encoding Scheme with Introduction of Bit Stuffing for efficient Data Compression

Asjad Amin, Haseeb Ahmad Qureshi, Muhammad Junaid, Muhammad Yasir Habib, Waqas Anjum
Department of Telecommunication and Electronic Engineering
The Islamia University of Bahawalpur, Pakistan

[asjad.amin@iub.edu.pk,haseebahmad89@gmail.com, luckyguy_juni89@yahoo.com, yasirhabib_gmi@yahoo.com, waqas.anjum@iub.edu.pk]

***Abstract*—This paper presents a modified scheme for run length encoding. A significant improvement in compression ratio for almost any kind of data can be achieved by the proposed scheme. All the limitations and problems in the original run length encoding scheme have been highlighted and discussed in detail in this research paper. A proposed solution has been suggested and performed for each problem to achieve intelligent and efficient coding. One of the major problems with original design is that a larger number of bits are used to represent length of each run. This has been resolved by introducing bit stuffing in RLE. Such larger sequences that affects compression ratio are broken into small sequences using bit stuffing. To allow more compression and flexibility, the length of maximum allowable bit sequence is not fixed and can be adjusted with input. Secondly we ignore the large numbers of small sequences that are largely responsible for expansion of data instead of compression. Four random sequences have been analyzed and when applied by modified scheme, a compression ratio of as high as 50% is observed.**

I. INTRODUCTION

Data compression is a process that reduces the amount of data in order to reduce data transmitted and decreases transfer time because the size of the data is reduced [1]. Data compression is commonly used in modern database systems. Compression can be utilized for different reasons including: 1) Reducing storage/archival costs, which is particularly important for large data warehouses 2) Improving query workload performance by reducing the I/O costs [2].

Data compression involves transforming a string of characters in some representation (such as ASCII) into a new string which contains the same information but with smallest possible length. Data compression has important application in the areas of data transmission and data storage. Compressing data reduces storage and communication costs. Similarly, compressing a file to half of its original size is equivalent to doubling the capacity of the storage medium. Data compression is rapidly becoming a standard component of communications hardware and data storage devices [3].

The paper is organized as follows: Section II presents the Original Run length encoding scheme, Section III presents modified run length encoding scheme to overcome the problems, Section IV verifies the result of proposed encoding scheme for four randomly chosen inputs. Section V presents Conclusion remarks.

II. RUN LENGTH ENCODING

Run-length encoding (RLE) is a very simple form of data compression in which runs of data (that is, sequences in which the same data value occurs in many consecutive data elements) are stored as a single data value and count, rather than as the original run. This is most useful on data that contains many such runs: for example, simple graphic images such as icons, line drawings, and animations. It is not useful with files that don't have many runs as it could greatly increase the file size.

The RLE algorithm performs a lossless compression of input data based on sequences of identical values (runs). It is a historical technique, originally exploited by fax machine and later adopted in image processing. The algorithm is quite easy: each run, instead of being represented explicitly, is translated by the encoding algorithm in a pair (l,v) where l is the length of the run and v is the value of the run elements. The longer the run in the sequence to be compressed, the better is the compression ratio [4].

A. Working of Run Length Encoding

An n bit of data is compressed by arranging it in the form of run and the count of each run. The count of each run is then represented in binary for the case of binary data. Amount of data compressed is directly related to length and number of longer runs. Run-length encoding when applied on data with information bits '11111111111100000000000000000011111" gives us a subset of 3 pairs, each pair representing number of runs and the bit (No of times bit occur, Bit). Hence, the above mentioned bit pattern in the run length encoding scheme is represented as (13,1)(17,0)(5,1). In binary form the latter pairs are expressed as follows:13=01101, 17=10001 and 5=00101. The final output comes out to be0110**1**1000**1**0**0**01011**.** In this way, the original pattern of n bits can be compressed to a great extent thereby reducing the data.

This encoding scheme does not always perform data compression. In some scenarios where the runs of smaller length are in excess, this scheme performs poorly and instead of compressing the data, the resultant output is an expanded form of the input. Consider a pattern "101010" when applied by Run Length Encoding, the final output comes out to be an expanded form of input data. The final output is (1,1)(1,0)(1,1)(1,0)(1,1)(1,0)or 111011101110 which is larger in size than the input.

In case of large consecutive runs of 1's or 0's, RLE performs efficient compression whereas in case of a data with large number of single 0's or 1's, the output is an expanded form of input sometimes the output is twice the size of input. This expansion of data instead of compression proves RLE technique less reliable. That is why run length encoding is a poor technique and practically not efficient for

larger data. The core objective of this research paper is to improve this encoding technique.

The length of largest run decides the number of bits needed to represent the count or length of each run in run length encoding technique. Consider a bit pattern 01010011111111111. The largest run in the data is 11 number of 1's appearing consecutively. Therefore this run decides the number of bits needed to represent length of each run in data. The number of bits needed to represent the length of run is 4 or the given scenario. The above sequence is written in run length encoding as:

TABLE I. RUN LENGTH ENCODING PAIRS FOR ABOVE DATA

BIT	RUN LENTH ENCODING
0	0001,0
1	0001,1
0	0001,0
1	0001,1
00	0010,0
11111111111	1011,1

B. Problem with Run Length Encoding

There are two basic problems that degrade the performance of Run Length encoding schemes.

Most of the Bits in a data are arranged in runs of smaller lengths that include single zeroes or single ones, double zeroes or double ones. Such combination requires more number of bits than their actual size to represent them in run length encoding technique. A single 0 may be represented as (000001, 0) which is seven times larger than the input data. This is one of the major performances limiting factor and results in expansion of data instead of compression.

Sometimes a data may contain a very large sequence of consecutive ones or zeros. Such sequences are represented in fewer numbers of bits in RLE but they might affect the overall compression in a negative way as largest sequence decides the number of bits to represent the length of a run in each pair. As a result the length of run in all the other sequences is also represented by the same number of bits for which the largest run is represented. For the scenario given below we need 4 extra bits to represent the length each single bit 0/1 because the longest sequence is represented by 4 bits as shown below

0	0001,0
111111111111111	1111,1

III. MODIFIED RUN LENGTH ENCODING SCHEME

A. Proposed Solution

As highlighted above, there are two very clear problems that decrease the performance of run length encoding scheme. We have proposed some modifications in run length encoding scheme. These modifications are specially designed to counter the above mentioned problems. The modified run length encoding scheme gives a significant improvement in compression ratio for almost any kind of data.

The above mentioned problem in Run Length Encoding Scheme can be overcome by intelligent compression. Analyzing the input data is the first and core step. We analyze data to highlight if there are any largest numbers of sequences that may increase the number of bits to represent the length of each run. Secondly we highlight the smaller sequences of single zeros/ones double zeros/ones or triple zeros/ones that may result in expansion of data instead of compression.

B. Bit Stuffing:

In data transmission bit stuffing is the insertion of non-information bits into data. Stuffed bits should not be confused with overhead bits. Bit stuffing is used for various purposes, such as for bringing bit streams that do not necessarily have the same or rationally related bit rates up to a common rate, or to fill buffers or frames. The location of the stuffing bits is communicated to the receiving end of the data link, where these extra bits are removed to return the bit streams to their original bit rates or form [5].

In run length encoding, bit stuffing is used to limit the number of consecutive bits of the same value in the data to be transmitted. A bit of the opposite value is inserted after the maximum allowed number of consecutive bits. Since this is a general rule the receiver doesn't need extra information about the location of the stuffing bits in order to do the de-stuffing. The receiver will only require the number maximum allowable consecutive bits.

We first break the large sequences that are usually very fewer in number but results in increasing the number of bits needed to represent sequence length. We break such sequences by introducing bit stuffing in RLE.15 consecutive ones are represented by 4 bits and 17 consecutive ones are represented by 5 bits. Therefore stuffing a zero after 15 ones in a17 bit sequence will break the overall sequence into two parts. This will limit the largest sequence to 15 consecutive ones and we will need only 4 bits to represent the length of each sequence instead of 5 bits. Length of a sequence after which a bit will be stuff is not fixed. For the above case it is taken as 15 but it can be adjusted as per the distribution of bits in a specific data. A single zero will be used to break a sequence of consecutive ones and a single one will be used to break a sequence of consecutive zeros.

C. Leaving small sequences out of RLE

All the smaller sequences that may result in expansion of data are kept out of RLE. We ignore the single zeros/ones, double zeros/ones that contribute in expanding data. Such sequences are left untouched and run length encoding scheme is not applied on them. The v in (l,v) or value of a run is not a single zero or one as a single zero or one will be mixed with the ignored sequences. We take the value of v as the smallest consecutive sequence that is considered for RLE. Consider a data 0101010000001111111. If we apply modified scheme and ignore single or double one/zeros. The output comes out to be 0101(110,**000**)(111,**111**). The value of a run for the above case will always be 000 for a sequence of zero bits and 111 for a sequence of one bit. This is

because we have ignored single and double one/zeros and the smallest sequence that is included in RLE is 000 or 111.

IV. VERIFYING MODIFIED RLE RESULTS

We have taken different random input sequences to verify our algorithm's results. These sequences are shown in figure 1, 2, 3and4. We can fairly analyze the data and highlight the problems with the help of given figures. Each figure represents the number of times a consecutive bit sequences is appearing in an input.

A. Step I, Analyzing Input Data

It is clear that a single zero/one or double zero/one occurs more than any other bit sequence. It can be verified by all the four figures. It can also be seen that some very larger sequences occur in almost every input data. These sequences are very fewer in numbers but even a single such sequence is enough to increase the number of bits to represent each data in case of original run length encoding.

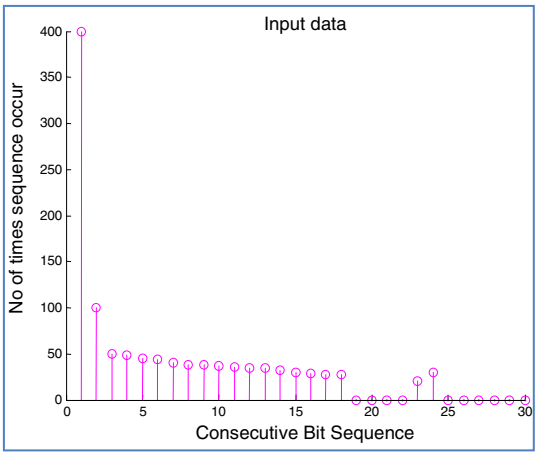

Figure 1. Distribution of consecutive bit sequences for Input 1

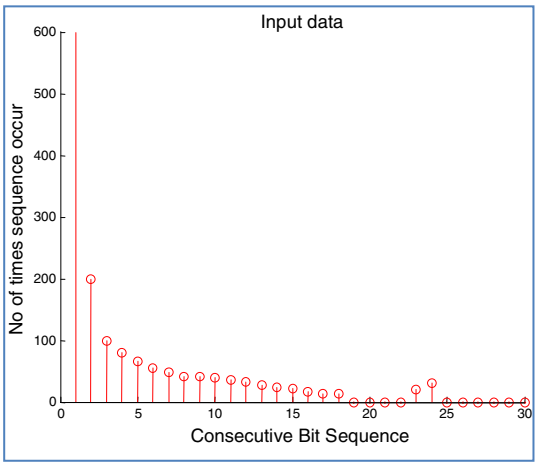

Figure 2. Distribution of consecutive bit sequences for Input 2

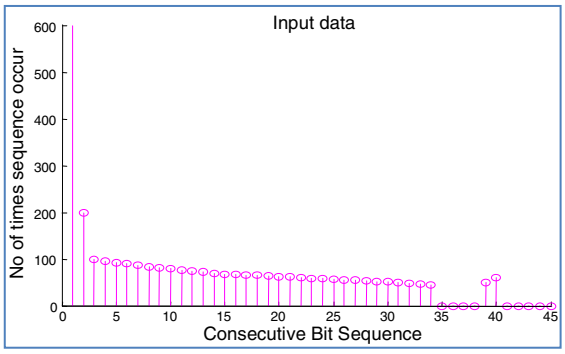

Figure 3. Distribution of consecutive bit sequences for Input 3

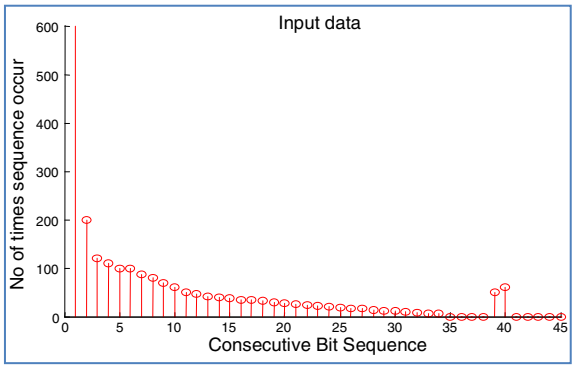

Figure 4. Distribution of consecutive bit sequences for Input 4

B. Step II, Bit stuffing to break lager sequences

We get rid of larger sequences by applying bit stuffing. The maximum allowable consecutive bits are different for each case. It depends on the distribution of input. In figure 5 we break all the sequences greater than 18 consecutive 0's/1's as such sequences are very fewer in number and will result in increasing the number of bits used to represent the length of every sequence. Breaking the sequence greater then 18 bits, as shown in figure 6, helps us representing the length of each sequence in 4 bits instead of 5. As we ignore as single zero/one and double zero/one. Therefore the sequences are 3-18 and count can be represented by 4 bits. For the case of figure 7 and 8 the maximum allowable consecutive bits is taken as 34 to limit the number of count bits to 5. Bit stuffed sequence for the above input data is shown in figure 5, 6, 7 and 8.

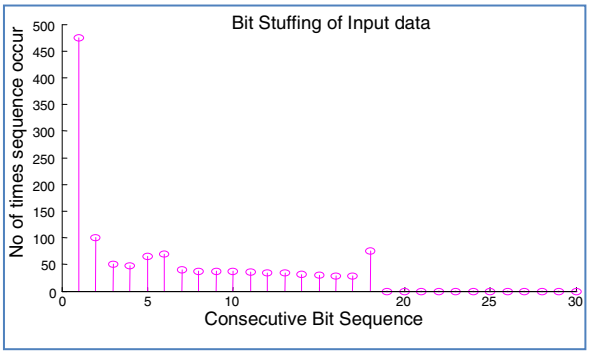

Figure 5. Data of Input 1 after Bit Stuffing

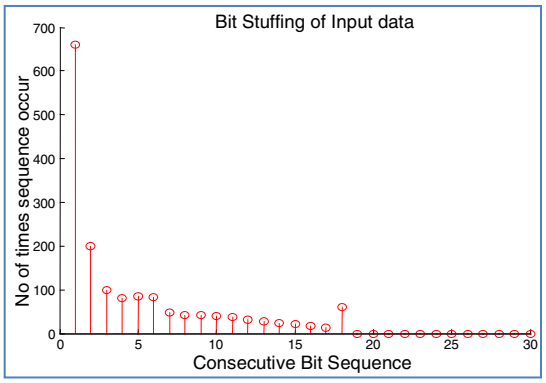

Figure 6. Data of Input 2 after Bit Stuffing

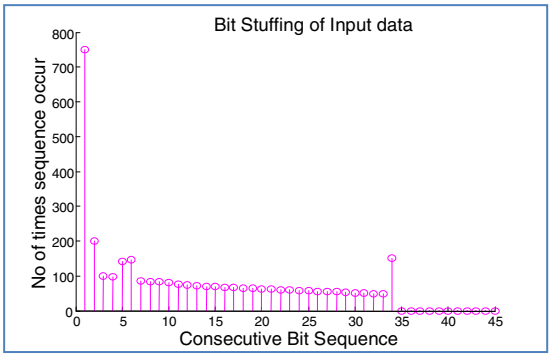

Figure 7. Data of Input 3 after Bit Stuffing

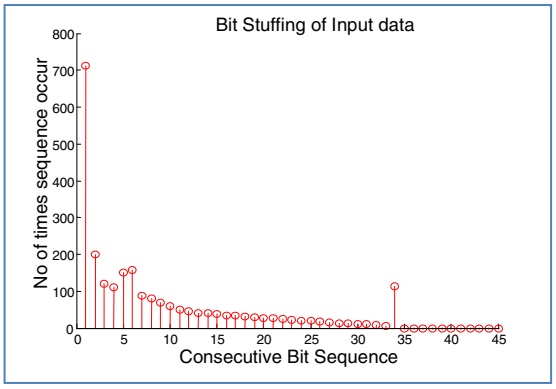

Figure 8. Data of Input 4 after Bit Stuffing

C. Step III, Ignoring small runs and applying modified RLE

In the last step, we ignore single 0's/1's and double 0's/1's and apply the Run Length Encoding Scheme on the remaining data. The data for the above inputs after bit stuffing and ignoring single 0/1 and double 0/1 is shown in figure 9, 10, 11 and 12. We observe that this data does not include any larger sequences that may effect the compression. This data also does not contain any large number smaller sequences that result in expansion of data. Therefore when RLE is applied on such data, we get a remarkable amount of improvement in compression.

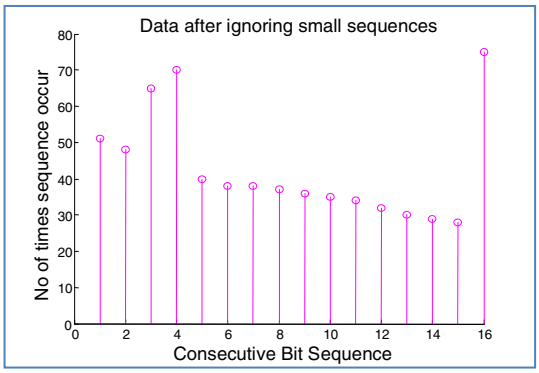

Figure 9. Data of Input 1 after bit stuffing & ignoring small sequences

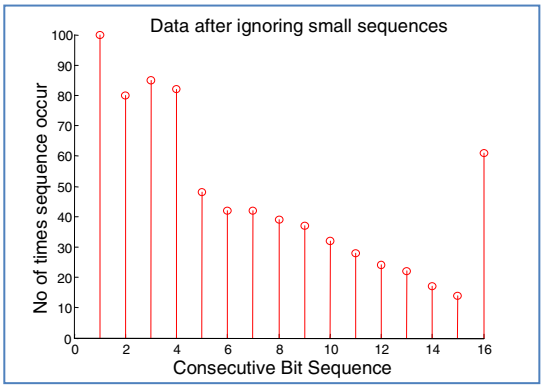

Figure 10. Data of Input 2 after bit stuffing & ignoring small sequences

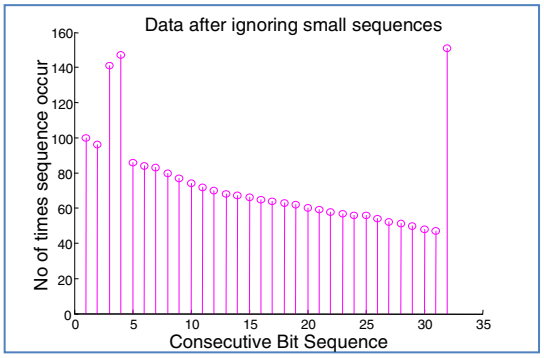

Figure 11. Data of Input 3 after bit stuffing & ignoring small sequences

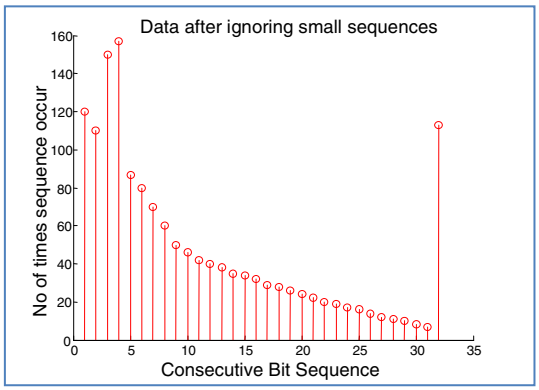

Figure 12. Data of Input 4 after bit stuffing & ignoring small sequences

Figure 13 and figure 14 shows the amount of compression that has been achieved using modified run length encoding for five randomly chosen input sequences, four of which are shown previously in figure 1, 2, 3 and 4. Figure 15 shows a comparative chart of original and compressed data.

Input Sequence	1	2	3	4	5
Total Bits	6327	6287	36488	16455	23381
Bits Saved	1583	681	18280	4959	2333
% of Data Saved	25	10.8	50.1	30.1	10

Figure 13. TCP Frame with Data segment divided into cells

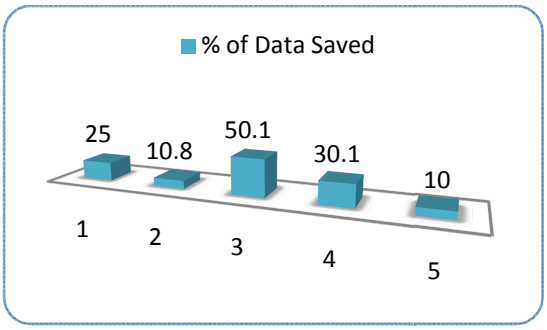

Figure 14. TCP Frame with Data segment divided into cells

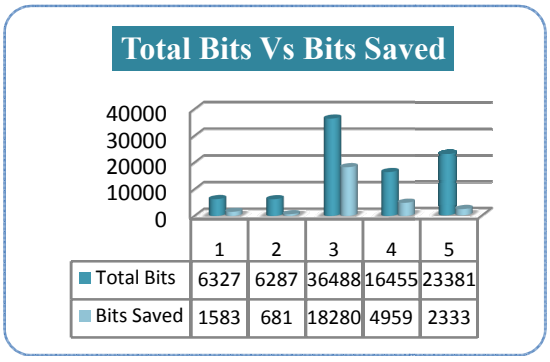

Figure 15. TCP Frame with Data segment divided into cells

At the receiver end, the output data is then fed as an input and same steps are performed in reverse order (i.e. in sequence of steps III, II & I) to extract the same data that was originally used as an input in Step I.

V. CONCLUSION

This research paper provides a new and more reliable technique for data compression. It solves the limitations present in Run Length Encoding Scheme. Problems in traditional run length encoding are highlighted and discussed in detail. A solution to each problem is then proposed in modified run length encoding scheme. Four random input sequences are taken and analyzed. To make RLE work we first use bit stuffing to break larger sequences and then ignore single 0's/1's and double 0's/1's respectively. Intelligent coding is done on the remaining data. Then, a combination of ignored single and double 0's/1's with run length encoded data is sent to receiver. The receiver applies all the steps in reverse order. A receiver applies a run length decoding scheme followed by de-stuffing of bits. The original data is then recovered at the receiver end. In this way expansion of data can be avoided in cases where Run Length Encoding fails. Our results show a compression of 50% to 10%.Even in worse scenario the modified algorithm will not expand the input.

REFERENCES

[1] Eug`enePamba Capo-Chichi, Herv´eGuyennet, Jean-Michel Friedt, "A new Data Compression Algorithm forWireless Sensor Network," in *Proc Third International Conference on Sensor Technologies and Applications,*2009, pp.1-6 DOI 10.1109/SENSORCOMM.2009.84

[2] StratosIdreos, RaghavKaushik, VivekNarasayya, Ravishankar Ramamurthy, "Estimating the Compression Fraction of an Index using Sampling,"in *Proc. International Conference on Data Engineering (ICDE)*, 2010, doi. 10.1109/ICDE.2010.5447694

[3] James A. Storer, "Data Compression Methods and Theory," Computer Science Press, 1988, 413 pp, ISBN-10: 0716781565

[4] Stefano Ferilli, "Automatic Digital Document Processing and Management: Problems, Algorithms and techniques,"ISBN: 0857291971

[5] Martin H. Weik, "Computer Science and Communications Dictionary,"2000, Volume 1, p.129