International Journal of Computer and Communication Technology

Volume 4 | Issue 2 Article 10

April 2013

A modified approach to the new lossless data Compression method

Bitla Srinivas
Department of Computer Science and Engineering National Institute of Technology Calicut Calicut, Kerala 673601, srinivas112@gmail.com

V.K. Govindan
National Institute of Technology Calicut Calicut, India, vkg@nitc.ac.in

Recommended Citation
Srinivas, Bitla and Govindan, V.K. (2013) "A modified approach to the new lossless data Compression method," *International Journal of Computer and Communication Technology*: Vol. 4 : Iss. 2 , Article 10.
DOI: 10.47893/IJCCT.2013.1181
Available at: https://www.interscience.in/ijcct/vol4/iss2/10

A modified approach to the new lossless data Compression method

Bitla Srinivas
Department of Computer Science and Engineering
National Institute of Technology Calicut
Calicut, Kerala 673601.
Email: srinivas112@gmail.com

V K Govindan
Department of Computer Science and Engineering
National Institute of Technology Calicut
Calicut, Kerala 673601.
Email: vkg@nitc.ac.in

***Abstract*— This paper proposed the modified approach to the new lossless data compression method. It is developed based on hiding data reversibly with a location map. It performs same as the earlier algorithm but it stands on lossless strategy where as the former approach could not do it. It can compress any kind of symbols as it operates on binary symbols. It is faster than many algorithms as it does not have any complex mathematical operations. Experimental results proved that when the symbol probability increases the algorithm shows good compression ratio.**

Keywords-Lossless data compression; Reversible data hiding

I. INTRODUCTION

Data compression is a necessary technique in the digital world, since storage capacity is always limited and we desire faster transfer of data over the network. In such a case we need to compress the data so that it can be recovered later. There exists two types of data compression algorithms. Lossless algorithms, which can reconstruct the original message exactly from compressed message, and lossy algorithms, which can reconstruct the approximation of the original message. Model and coder are two components in compression for capturing probability distribution of input message and generating code based on those probability information respectively. Information theory ties model and coder together. A related key term in this context is entropy. Entropy is defined as the smallest number of bits needed to represent each symbol on an average. For a set of possible message *S*, first order entropy is given by

$$H(S) = \sum_{s \in S} p(s) \log_2 \left(\frac{1}{p(s)}\right) \qquad (1)$$

where *p(s)* is the probability of message *s*.

For Lossless data compression, entropy coding and dictionary coding algorithms are the two main stream of techniques. Huffman coding [1] and arithmetic coding [2] are good examples of entropy coding. Lempel-Ziv [3] or Lempel-Ziv Welch [4] algorithms are examples of dictionary coding that builds dictionary. Quality of lossless data compression depends on some criteria like the time to compress, time to reconstruct, the size of the compressed message and generality. Performance can be measured by entropy and compression ratios. Whenever average length of the code nearly equal to the entropy then we can say that corresponding algorithm is a better compression algorithm.

$$Compression\ ratio = \frac{Compressed\ size}{Uncompressed\ size} \qquad (2)$$

It would be less than 1 for better compression algorithm.

Reversible data hiding [5] is also known as lossless embedding, which enables the exact recovery of the original host message upon extraction of the embedded information. Thus, reversible data hiding techniques are frequently used to prove authenticity of transmitted data. This reversible data hiding technique is used in lossless data compression by Hyoung Joong Kim [6]. Occurrence of one symbol is more than that of occurrence of another symbol then we can say that first symbol is highly skewed than second symbol. The method in [6] can compress data with highly skewed symbol probability and also compress the data that is already compressed by unbalanced Huffman tree. Under certain condition, the method [6] can further compress the compressed data.

II. RELATED WORK

Many algorithms are defined in lossless data compression.

Shannon-fano coding [7] was the first method developed for finding good variable-size codes. In this first symbols are arranged in descending order of their probabilities. These are divided into two parts with equal sum of their probabilities. First part coding starts with 0, where as the second part coding starts with 1. After that, these 2 parts can be subdivided and coded repeatedly in the same way. This will continue until no more divisions possible.

Huffman's algorithm [7], [1] has provided solution to the problem of constructing minimum redundancy code. It compresses by transmitting the more probable symbol with less number of bits than less probable symbol. Huffman codes are properly decoded because they obey prefix property, which means that no code word can be a prefix of another code word. Complete set of code words can be represented as a binary tree, known as a Huffman tree.

Huffman coding is widely used in many applications such as baseline compression standards including JPEG, MPEG for audio, video, and text. Huffman coder must use at least one bit per symbol. Extended Huffman coding can achieve fractional bits per symbol by blocking several symbols together at a time [8].

Arithmetic coding [7], [2] transmits a string into a code so that it can achieve the fractional bits to represent a symbol. It is particularly useful when dealing with highly skewed symbols in probability [9].

The simplest coding that takes the advantage of the context is run-length coding [7]. The basic idea is to identify symbols of same value which present adjacently and replace them with a single occurrence along with a count. Move-to-front coding [7] also takes the advantage of context. The idea of its coding is to preprocess the message sequence by converting it into a sequence of integers.

PPM (Prediction by Partial Matching) [7] is to take the advantage of previous k characters to generate a conditional probability of current character. The probability distributions can then be used by a Huffman or Arithmetic coder to generate bit sequence.

Lempel-Ziv algorithms [3], [4] compress by building a dictionary of previously seen strings. LZ77 algorithm [10] uses a sliding window that moves along with the cursor. It looks through the preceding part of the given string to find the longest match to the string starting at the current position and outputs the code that refers that match. Eg: ZIP, Gzip. LZ78 algorithm [4] is based on a more conservative approach to add the string to the dictionary. Eg: Unix compress and Gifs. Burrows Wheeler algorithm [11] is one of the best overall compression algorithms. It avoids the problems of PPM in selecting k. It uses reverse lexical order sorting with move-to-front coding [7]. Characters with the similar contexts (preceding characters) are near each other. Longer match can be detected fast. The process of sorting the characters by their context is often referred to as a block-sorting algorithm.

A new lossless data compression method [6] proposed by Hyoung Joong Kim, which is completely different from Huffman coding [1] and Arithmetic coding [2]. It is based on skewed probability of symbols. Maintaining tree or table is not necessary. It needs to count the number of zeros and ones of the given bit string. It uses reversible data hiding [5] technique for compressing. This method does not need to send information to the decoder about probability of symbols. It divides the string into two parts: U and V . After compressing with this method, the resultant parts are L and M.

The Illustrative examples given in [6] describe the working of the algorithm. The given input string X is divided into U and V based on probability of highest skewed symbols. The division is done in such a way that the length of U is more than length of V mostly. When 0s are more skewed than 1s, the length of U is given by

$$|U| = \frac{|X|}{1 + P0} \qquad (3)$$

where P_0 is the probability of 0 in the input string X.

If 1s are skewed than 0s then the following principle is used

$$|U| = \frac{|X|}{1 + P1} \qquad (4)$$

Where, P_1 is the probability of 1s in the input string X. It is observed that the flag bit used in the above algorithm [6] to handle the last bit padded to L introduces new problem as explained in the following example.

Consider example 1 [6], given an input string X = {10100000000010000010} with $|X|$=21. X contains 17 0s and 4 1s. It can be divided into U and V based on above principle we will get first part as U with $|U|$=11 and second part as V with $|V|$=10. In compressing process, out of 10 symbols of V only 9 can be embeddable in the 0s of U. The last unused bit of V is appended to the embedded result L and f is set to true. The resultant strings are L = {101010000010}, M ={0010} and f = 1 as shown in fig.1.

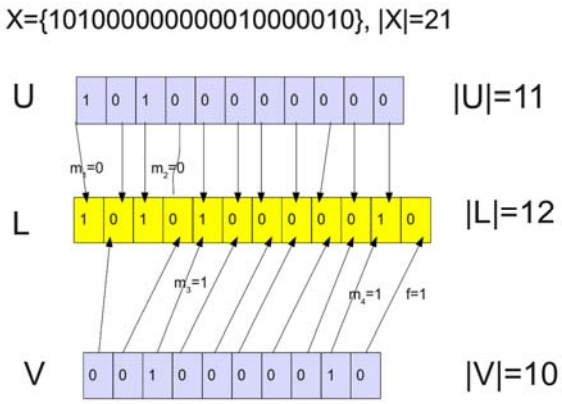

Fig. 1. Embedding result of U and V

Coming to the decompression, flag bit 0 means that extract the data from L at last bit. If flag bit is 1, it indicates that there is no data to extract from last bit and append to U or V. But this algorithm didn't mention to which last bit has to append whether U or V as shown in fig.2.

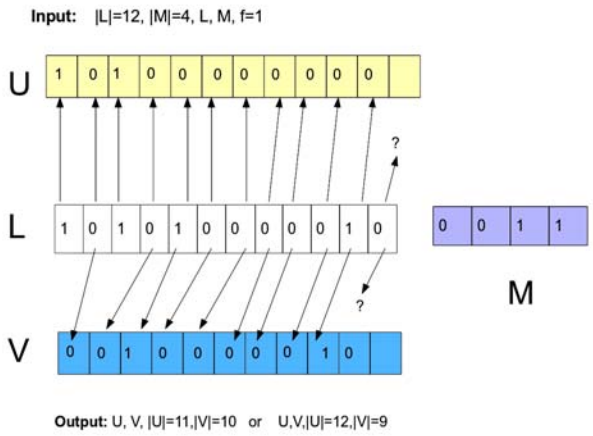

Fig. 2. Recovering data from embedded result *L*

The compression algorithm [6] clearly stated that only last bit of *L* left in dilemma whether to recover data from it or not. But there are some other bits apart from last bit when recovers, lead to undesirable results. For example 2, consider *X* is {10100000000001000001000000100101001010010010000000000000000000000000000010} with |*X*| = 74, calculated |*U*| and |*V*| are 39 and 35 respectively. After compressing by the method [6] the resultant *L* is {11100000000001000001000000100101001010000010}, *M* is {0100000000}, |*L*| = 44, |*M*| = 10 and *f* = 1. But decompression side, results are |*U*| = 43; |*V* | = 35 as shown in fig.3.

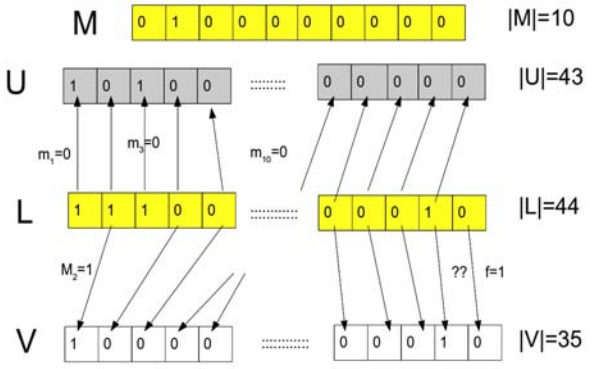

Fig. 3. Recovering the data from *L* with *M*

Reason is last 4 bits of *L* are inserted without embedding. With this example the compression method [6] is proved as a lossy compression method. We need to know how much extent, the data is hidden in *L* bits to make it as lossless compression method.

Some times the algorithm [6] resulting negative compression ratio even though when bits are skewed one upon other. For example 3, consider *X* is {10111100110000100000000100} with |*X*| = 25, calculated |*U*| and |*V*| are 15 and 10 respectively. After compressing the result *L* is {101111001100001100}, *M* is {00000000}, |*L*| = 18 , |*M*| = 8 and *f* = 1. Compression ratio is 1.040000. In the given string 0s more skewed on 1s. Bits of *V* is to hide in 0s of *U*, but most of the 1s of *X* present in *U* only. So embedding all bits of *V* is not possible and for every 1 in embedded result corresponding *M* bit will be there.

III. MODIFIED LOSSLESS DATA COMPRESSION METHOD

In the existing lossless data compression method [6] header contains |*M*|, *M*, *f* and |*L*|. |*M*| and |*L*| are used in modified method as well. The String *M* is used to decode the String *L* with same rule as in method [6]. The bit *f* introduces problem of placing last bit of *L* as mentioned in the example 1 and we need to know exact length of *L* up to which embedding process has done with *U* and *V*. So it is considerable to include embedded string length *Ebl* in header design. Value of *Ebl* is always less than or equal to |*L*|. There are unused bits of *L* which are appended from *U* or *V* directly. To differentiate those unused bits we use the flag bit *f*. Flag bit *f* is 0, indicates to decoder that unused bits came from *U*. Flag bit *f* is 1, indicates to decoder that unused bits came from V. Decoder looks at flag bit *f* only when *Ebl* is less than |*L*|. And we make sure that |*U*| always greater than |*V*| using equations (3) and (4).

A. Compression Algorithm

Input: Bit string *X*
Output: Resultant String *L* and *M*, Header information is |*L*| and |*M*|, Embed length *Ebl* and flag *f*.

Initialize *i*, *j*, *k*, *Ebl* to 0
Divide *X* into *U*, *V* such that |*U*| is greater than |*V*| using (3) and (4)
for each *i* till U_i is invalid **do**
  if V_j is valid **then**
    if U_i is 1 **then**
      $L_i \leftarrow 1$
      $M_k \leftarrow 0$
      increment *k* by 1
    else
      $L_i \leftarrow V_j$
      if V_j is not 0 **then**
        $M_k \leftarrow 1$
        increment *k* by 1
      end if
      increment *j* by 1
    end if
  else
    exit **for** loop
end for
Ebl ← *i*
while U_i is valid **do**
  f ← 0
  $L_i \leftarrow U_i$

```
        increment i by 1
end while
while V_j is valid do
    f ←1
    L_i ← V_j
    increment i by 1
    increment j by 1
end while
```

B. Decompression Algorithm

Input: Header includes embed limit *Ebl,* flag bit *f,* length of *M* |*M* |, length of String *L* |*L*| and Strings *M*, input String *L*.
Output: Bit String *X*

```
Initialize i, j, k, m to 0
for each i till i is Ebl do
    if L_i is 0 then
        U_j ← 0
        increment j by 1
        V_k ← 0
        increment k by 1
    else
        if M_m is 0 then
            U_j ← 1
            increment j by 1
            increment m by 1
        else
            V_k ← 1
            increment k by 1
            U_j ← 0
            increment j by 1
            increment m by 1
        end if
    end if
end for
if Ebl is less than |L| then
    if f is 0 then
        while L_i is valid do
            U_j ← L_i
            increment i by 1
            increment j by 1
        end while
    else
        while L_i is valid do
            V_k ← L_i
            increment i by 1
            increment k by 1
        end while
    end if
end if
    X ← Concatenation(U, V )
```

These algorithms are applicable when zeros more than ones. If the number of ones dominates zeros, it can be applied after flipping ones and zeros.

IV. PERFORMANCE COMPARISONS

For the experiment, we have chosen 118 binary symbols with different probabilities as shown in fig. 4. The proposed

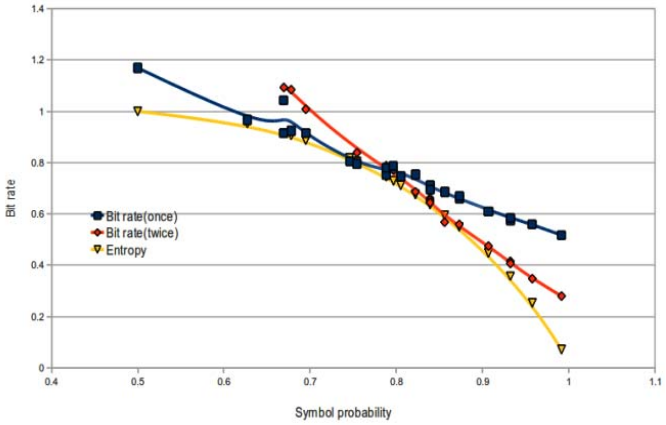

Fig. 4. Bitrate of proposed method using 118 binary symbols with different symbol probabilities.

method reaches near entropy at certain range. It observed that performance is linearly proportional to the probability. Moreover, the compression ratio depends on position of symbols present in the input string. We can observe one glitch represents negative compression even when skewness exists between symbols. When we increase the probability of symbols, bit-rate (one time compression) is widen from entropy curve. When the symbol probability is high, bit-rate (compression twice) is closer to the entropy. When compared with existing algorithm, this algorithm gives accurate results and it is purely lossless data compression algorithm. In this flag bit *f* is used in differently such that *f* is 0 to indicate padded bits are from *U,* and if *f* is 1 to indicate padded bits are from V and embed length *Ebl* is added in header design to know exact position up to where data is hidden in String *L*.

V. CONCLUSION

We have identified the drawbacks of new lossless data compression method [6]. We have modified the algorithm to make it stand on lossless strategy where the existing algorithm fails. The modification is carried out without affecting the speed performance of the algorithm [6]. This is the fastest algorithm because we do not have complex mathematical equations and trees, just we need to calculate the number of zeros and ones. It can be used for multimedia communications and future applications on lossless data compression. It is simple and works on any kind of symbols.

REFERENCES

[1] D. Huffman, "A method for the construction of minimum-redundancy codes," *Proceedings of the IRE*, vol. 40, no. 9, pp. 1098–1101, sep.1952.

[2] G. G. Langdon, "Arithmetic coding," *IBM J. Res. Develop*, vol. 23, pp.149–162, 1979.

[3] J. Ziv and A. Lempel, "Compression of individual sequences via variable-rate coding," *Information Theory, IEEE Transactions on,* vol. 24, no. 5, pp. 530–536, sep. 1978.

[4] T. A. Welch, "A technique for high-performance data compression," *Computer,* vol. 17, no. 6, pp. 8–19, 1984.

[5] M. Celik, G. Sharma, A. Tekalp, and E. Saber, "Reversible data hiding," *in Proceedings of IEEE International Conference on Image Processing*, vol. 2. Citeseer, 2002, pp. 157–160.

[6] H. J. Kim, "A new lossless data compression method," jun. 2009, pp.1740–1743.

[7] D. Salomon, Data Compression: *The Complete Reference*, 2nd ed., 2004. [Online]. Available: http://www.ecs.csun.edu/ dxs/DC3advertis/Dcomp3Ad.html

[8] K. Sayood, *Introduction to data compression*. Morgan Kaufmann, 2000.

[9] I. Witten, R. Neal, and J. Cleary, "Arithmetic coding for data compression," *Communications of the ACM*, vol. 30, no. 6, pp. 520–540, 1987.

[10] A. Lempel, "A universal algorithm for sequential data compression," *IEEE Transactions On Information Theory*, vol. 23, no. 3, pp. 337-343, 1977.

[11] A. B. sorting Lossless, M. Burrows, M. Burrows, D. Wheeler, and D.J. Wheeler, "A block-sorting lossless data compression algorithm," Digital SRC Research Report, Tech. Rep., 1994.