# Rearrangement and Grouping of Data Bits for Efficient Lossless Encoding

**Ajitha Shenoy K B[1] , Meghana Ajith[2] and Vinayak M Mantoor[3]**

[1, 3]Department of Information and Communication Technology, Manipal Institute of Technology, Manipal University, Manipal, Karnataka, India
[2] Department of Computer Applications, Manipal Institute of Technology, Manipal University, Manipal, Karnataka, India

E-mail: ajith.shenoy@manipal.edu, meghana.ajith@manipal.edu, vinayak.mantoor@manipal.edu

**Abstract**. This paper describes the efficacy of rearranging and grouping of data bits. Lossless encoding techniques like Huffman Coding, Arithmetic Coding etc., works well on data which contains redundant information. The idea behind these techniques is to encode more frequently occurring symbols with less number of bits and more seldom occurring symbols with more number of bits. Most of the methods fail if there is a non-redundant data. We propose a method to re arrange and group data bits there by making the data redundant and then different lossless encoding techniques can be applied. In this paper we propose three different methods to rearrange the data bits, and efficient way of grouping them. This is first such attempt. We also justify the need of rearranging and grouping data bits for efficient lossless encoding.

## 1. Introduction

Data compression is important in the field of information theory and signal processing. Good encoding techniques reduce amount of memory needed to store the large file and also make transmission of such file over network faster. Good encoding implies efficient way of utilizing computer memory and network bandwidth. There are many encoding techniques defined in the literature. Compression techniques are classified into three categories. Loss less compression, Lossy compression and hybrid compression. Well known lossless compression techniques are run-length coding, Shannon-Fano Coding [1], Huffman coding [2], LZW encoding [3, 4], arithmetic coding [5, 6], lossless predictive encoding etc.[1]. Lossy compression techniques are discrete cosine transformation, KL transformation and lossy predictive encoding, differential pulse code modulation, delta modulation, wavelet transform etc[1]. Hybrid encoding techniques are combination of lossy and lossless encoding techniques. Examples are JPEG, MPEG, DVI, H.26X etc[1]. M. Kafashan et.al. in their paper proposed new rectangular partitioning methods for lossless binary image compression. They proposed 2D RLE which is efficient than 1D RLE scheme [7]. In this paper our work focuses on lossless compression techniques. Most of the lossless encoding techniques rely on frequency of occurrence of symbols in the given document. If all the symbols in the given document are distinct then all the lossless compression techniques like Huffman, coding, arithmetic coding, and predictive coding fail to encode it using fewer number of bits. We propose a new scheme which transforms the given data into redundant data so that lossless compression can be applied efficiently. In the next section we give definition and concepts which are needed to understand our work described in this paper. The rest of the paper is organized as follows. Section 2 gives basic concepts and definitions used in our work. Section 3 gives motivation for rearranging and grouping, Section 4 describe our proposed work and last section contains conclusion and future work.

## 2. Preliminaries

Entropy plays important role in lossless compression. Famous scientist Claude E. Shannon of Bell Labs defined entropy of information source as follows:

**Definition 2.1 (Entropy)** [1, 8, 9]: Entropy $\eta$ of an information source with alphabets S = { $s_1$, $s_2$, …, $s_n$ } is defined as $\eta = \Sigma^{n}_{i=1} P_i \log_2 (1/P_i)$, where $P_i$ denote probability of occurrence of symbol $s_i$ in given information.

Entropy denotes measure of disorder in a system. More entropy implies more disorder. As far as compression is concerned, entropy denote minimum average number of bits needed to represent each symbol $s_i$ in S. It specifies the lower bound for the average number of bits needed to code each symbol $s_i$ in S. Any lossless compression technique is good if average number of bits need to encode each symbol in the given information is approximately equal or equivalent to the entropy measure $\eta$. So entropy is very important measure to compare lossless compression techniques. Most widely used lossless encoding techniques are Huffman coding, Arithmetic coding and Run length coding. All these coding techniques are based on variable length coding. Main key idea in these compression techniques are: Frequently occurring characters are encoded with lower number of bits and seldom occurring characters are coded with higher number of bits [1]. Hence it is very much important that the given information should consist of redundant information so that these techniques can be applied on it efficiently.

**Definition 2.2 (Compression Ratio)**[1]: Compression ratio is defined as $B_0/B_1$, where $B_0$ denote number of bits needed to represent the given data before compression and $B_1$ denote number of bits needed to represent data after compression.

Note that more compression ratio implies more compression and vice versa. In the next section we describe our method of rearranging the given data so that it contains more redundant information and thus suitable for applying lossless encoding techniques on it efficiently.

## 3. Motivation

Let us justify the need of rearrangement of data. With the following example we will explain why rearrangement of data bits is important for lossless encoding.

**Example 2.1:** Let D = $D_0$. $D_1$, …,$D_{255}$ , where $D_i \neq D_j$ for $i \neq j$ and $D_i$ belong to set S={0, 1, …, 255} for all $0 \leq i \leq 255$.That is D is rearrangement of symbols in S in random order. Suppose we want to encode this using loss less compression technique like Huffman or Arithmetic coding, it will take 8 bit to represent each symbol in the information. Shannon's entropy measure also indicates that minimum average number of bits needed to encode each symbol in D requires 8 bits. So we can't do better than this. The only way to do better is rearrangement of data bits and grouping them in such a way that the grouped values are redundant and hence efficiently apply lossless compression techniques on grouped values. Example 2.1 motivates us to find rearrangement of data bits in such a way that, we can apply lossless encoding techniques efficiently on rearranged data.

## 4. Proposed Method

Suppose that the given data is D = $D_1$. $D_2$, …, $D_m$ , where $D_i$ be *n* - bit number for all $1 \leq i \leq m$. Now we explain rearrangement scheme for the given data D. Since each $D_j$ is n-bit number for all $0 \leq j \leq m$, we can write bit representation of each $D_i$ in given data D. So the entire data D can be expressed as *m x n* bit string. i.e.,

$D_1 = b_{11}, b_{12}$ …….. $b_{1n}$ $D_2 = b_{21}$ $b_{22}$ …….. $b_{2n}$ etc… $D_m = b_{m1}$ $b_{m2}$ …….. $b_{mn}$

Where, $D_i = b_{i1}$ $b_{i2}$ …….. $b_{in}$ represent n-bit representation of symbol $D_i$ in D. Now we define the method of rearranging data bits. We arrange data bits in three different types as follows:

**Type i:** Arrange m x n bit sequence in the given order without any change i.e. $b_{11}, b_{12}$ …….. $b_{1n}$ $b_{21}$ $b_{22}$ …….. $b_{2n}$ ………. $b_{m1}$ $b_{m2}$ …….. $b_{mn}$

**Type ii:.** Arrange the *m x n* bit sequence in the following manner.

MSB ($D_1$. $D_2$, …, $D_m$ ) Next MSB ($D_1$. $D_2$, …, $D_m$ )…….. LSB($D_1$. $D_2$, …, $D_m$ )

i.e. $b_{11}, b_{21}$ …….. $b_{m1}$ $b_{12}$ $b_{22}$ …….. $b_{m2}$ ………. $b_{1n}$ $b_{2n}$ …….. $b_{mn}$

**Type iii:** Consider each significant bit of data separately as independent bit streams.

Stream 1: MSB ($D_1$. $D_2$, …, $D_m$ ) = $b_{11}, b_{21}$ …….. $b_{m1}$

Stream 2: Next MSB ($D_1$. $D_2$, …, $D_m$ ) = $b_{12}$ $b_{22}$ …….. $b_{m2}$

…………………………………………… Stream n: LSB($D_1$. $D_2$, …, $D_m$ ) = $b_{1n}$ $b_{2n}$ …….. $b_{mn.}$

For the given input data, our aim is to find best suitable rearrangement and grouping of data bits. We group *(n + k) bits* (where k is varied from –n/2 to n/2) in each type of arrangements and calculate Shannon's entropy value η. Finally we select the rearrangement of bits which takes less entropy value. Less entropy implies less disorder in the given data. Now we formally state the algorithm for grouping bits in each arrangement and to select best rearranged data bits and best value of k, which is efficient for loss less encoding

**Algorithm: Finding the Best Rearrangement**

```
{   Type = NULL;
    For each of the Type i, Type ii do the following
    {   Best_k = 0;
        Best-Entropy = infinity;
        For k = – n/2, – n/2+1,…, 0, 1, 2, …,n/2 do
        {   Group n + k bits   // (from  m x n  bits we can get [m x n / (n + k) ] values)
            Convert decimal equivalent of each binary n + k  bits
            Find Shanon's entropy measure  η for  [m x n / (n + k) ] decimal values
            If (Best-Entropy > η) {   Best-Entropy = η
                                      Best_k = k
                                      Type = type i or ii for which
                                      Best_k is updated.}}}
    For Type iii do the following
    {   For each Stream i (i = 1, 2, ….n) do the following
        {   For k = – n/2, – n/2+1,…, 0, 1, 2, …,n/2 do
            {   Group n + k bits   of  Stream i.
                // (from  m   bits we can get [m / (n + k) ] values)
                Convert decimal equivalent of each binary n + k  bits
                Find Shanon's entropy measure  $η_i$ for  [m  / (n + k) ] grouped decimal values //  where $η_i$ denotes entropy of Stream i grouped values}
            Let N denote number of bits needed to represent entire data D, then
            $N = ( η_1 + η_2 + ..... + η_n) m/[n + k]$
            N1= Best_Entropy x  m x n/(n+k)
            // where N1 denote, number of bits needed to represent entire data using
            // best Rearrangement (Type) found so far in the previous for loop for
            // type i & ii
            If (N  <  N1 ) then  {Type = Type iii ; Best_k = k}}}
```

Algorithm 4. 1: Finding Best Rearranged Data Bits

Note that if *m* and *m x n* is not exactly divisible by (n + k) add trailing zero to the bit string and make it divisible by *(n + k)*. It is trivial to note that time complexity of above algorithm is *O(nm)*. Usually n is either 8 or 16 or 24 bits, hence algorithm takes linear time in m, i.e. O(m). Once we get Best-Entropy, type of rearranged data bits (type i or ii) and Best_k values, we can consider group of *(n + Best_k)* bits from that type and efficiently apply existing lossless encoding techniques on it. Note that decoder should know type of arrangement of data bits, Best_k, m and n values for decoding the data. These additional information should be sent to decoder to decode the data. Since decoder has the information about m, n and Best_k it can remove trailing zeroes( if any). Next we will give one example which shows why rearranging data bits and grouping it into n + k bits needed.

**Example 4.2:** Let D denote 256 consecutive numbers from 0, 1, 2, …. ,255.
Shanon's entropy measure for the given data D is $\eta = 8$ bits. Hence minimum average number of bits needed to represent each symbol in D is 8 bits. Therefore, Number of bits needed to represent D is at least 256 x 8 = 2048 bits. We consider rearrangement of data bits with respect to Type i, ii and iii. Let k = 0.

**Type i:** with Type i arrangements of data bits and k = 0, we get same data D as it is. Hence Entropy won't change ( $\eta = 8$ )

**Type ii** arrangement of data bits. We get,
(00…..0)$_{128\ \text{times}}$ (11….1)$_{128\ \text{times}}$ (00…..0)$_{64\ \text{times}}$ (11….1)$_{64\ \text{times}}$(00…..0)$_{64\ \text{times}}$ (11….1)$_{64\ \text{times}}$(00…..0)$_{32\ \text{times}}$ (11….1)$_{32\ \text{times}}$ (00…..0)$_{32\ \text{times}}$ (11….1)$_{32\ \text{times}}$(00…..0)$_{32\ \text{times}}$ (11….1)$_{32\ \text{times}}$ (00…..0)$_{32\ \text{times}}$ (11….1)$_{32\ \text{times}}$(00…..0)$_{16\ \text{times}}$ (11….1)$_{16\ \text{times}}$ (00…..0)$_{16\ \text{times}}$ (11….1)$_{16\ \text{times}}$ (00…..0)$_{16\ \text{times}}$ (11….1)$_{16\ \text{times}}$ (00…..0)$_{16\ \text{times}}$ (11….1)$_{16\ \text{times}}$(00…..0)$_{16\ \text{times}}$ (11….1)$_{16\ \text{times}}$ (00…..0)$_{16\ \text{times}}$ (11….1)$_{16\ \text{times}}$(00…..0)$_{16\ \text{times}}$ (11….1)$_{16\ \text{times}}$ (00…..0)$_{16\ \text{times}}$ (11….1)$_{16\ \text{times}}$ (00…..0)$_{8\ \text{times}}$ (11….1)$_{8\ \text{times}}$ (00…..0)$_{8\ \text{times}}$ (11….1)$_{8\ \text{times}}$(00…..0)$_{8\ \text{times}}$ (11….1)$_{8\ \text{times}}$ (00…..0)$_{8\ \text{times}}$ (11….1)$_{8\ \text{times}}$(00…..0)$_{8\ \text{times}}$ (11….1)$_{8\ \text{times}}$ (00…..0)$_{8\ \text{times}}$ (11….1)$_{8\ \text{times}}$(00…..0)$_{8\ \text{times}}$ (11….1)$_{8\ \text{times}}$ (00…..0)$_{8\ \text{times}}$ (11….1)$_{8\ \text{times}}$(00…..0)$_{8\ \text{times}}$ (11….1)$_{8\ \text{times}}$ (00…..0)$_{8\ \text{times}}$ (11….1)$_{8\ \text{times}}$(00…..0)$_{8\ \text{times}}$ (11….1)$_{8\ \text{times}}$ (00…..0)$_{8\ \text{times}}$ (11….1)$_{8\ \text{times}}$(00…..0)$_{8\ \text{times}}$ (11….1)$_{8\ \text{times}}$ (00…..0)$_{8\ \text{times}}$ (11….1)$_{8\ \text{times}}$ (00001111)……$_{32\ \text{times}}$……. (00001111) (00110011)……$_{32\ \text{times}}$……. (00110011) (01010101)……$_{32\ \text{times}}$……. (01010101)

Now grouping n + k bits (let us take k = 0 here for simplicity). i.e. grouping 8 bits we get
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 0, 0, 0, 0, 0, 0, 0, 0, 255, 255, 255, 255, 255, 255, 255, 0, 0, 0, 0, 0, 0, 0, 0, 255, 255, 255, 255, 255, 255, 255, 255, 0, 0, 0, 0, 255, 255, 255, 255, 0, 0, 0, 0, 255, 255, 255, 255, 0, 0, 0, 0, 255, 255, 255, 255, 0, 0, 0, 0, 255, 255, 255, 255, 0, 0, 255, 255, 0, 0, 255, 255, 0, 0, 255, 255, 0, 0, 255, 255, 0, 0, 255, 255, 0, 0, 255, 255, 0, 0, 255, 255, 0, 0, 255, 255, 0, 255, 0, 255, 0, 255, 0, 255, 0, 255, 0, 255, 0, 255, 0, 255, 0, 255, 0, 255, 0, 255, 0, 255, 0, 255, 0, 255, 0, 255, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 85, 85, 85, 85, 85, 85, 85, 85, 85, 85, 85, 85, 85, 85, 85, 85, 85, 85, 85, 85, 85, 85, 85, 85, 85, 85, 85, 85, 85, 85, 85, 85.

Now entropy of this new values $\eta_{new}$ = 1/256 [32 $\log_2$ (256/32) + 32 $\log_2$ (256/32) + 32 log $_2$(256/32) + 80 log $_2$(256/80) + 80 log $_2$(256/80)] = 2.17 Which is lesser than the previous $\eta$.

**Type iii :** By grouping 8 bits we get

**Stream 1:** 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255. Entropy $\eta_1 = 1$

**Stream 2:** 0, 0, 0, 0, 0, 0, 0, 0, 255, 255, 255, 255, 255, 255, 255, 255, 0, 0, 0, 0, 0, 0, 0, 0, 255, 255, 255, 255, 255, 255, 255, 255. Entropy $\eta_1 = 1$

**Stream 3:** 0, 0, 0, 0, 255, 255, 255, 255, 0, 0, 0, 0, 255, 255, 255, 255, 0, 0, 0, 0, 255, 255, 255, 255, 0, 0, 0, 0, 255, 255, 255, 255. Entropy $\eta_1 = 1$

**Stream 4:** 0, 0, 255, 255, 0, 0, 255, 255, 0, 0, 255, 255, 0, 0, 255, 255, 0, 0, 255, 255, 0, 0, 255, 255, 0, 0, 255, 255, 0, 0, 255, 255. Entropy $\eta_1 = 1$

**Stream 5:** 0, 255, 0, 255, 0, 255, 0, 255, 0, 255, 0, 255, 0, 255, 0, 255, 0, 255, 0, 255, 0, 255, 0, 255, 0, 255, 0, 255, 0, 255, 0, 255. Entropy $\eta_1 = 1$

**Stream 6:** 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15. Entropy $\eta_1 = 1$

**Stream 7:** 51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51. Entropy $\eta_1 = 1$

**Stream 8:** 85, 85, 85, 85, 85, 85, 85, 85, 85, 85, 85, 85, 85, 85, 85, 85, 85, 85, 85, 85, 85, 85, 85, 85, 85, 85, 85, 85, 85, 85, 85, 85. Entropy $\eta_1 = 1$

Hence total number of bits needed to represent entire data = (1 + 1 + 1+ 1+ 1 + 1 + 1 + 1)256/8 = 256 bits. Which is optimal possible compression with compression ratio = 2048/256 = 8 .

## 5. Conclusion and Future Work

In this paper we proposed a method for rearranging the data bits and grouping them. This is first such attempt. We defined three ways of rearranging the data bits and then grouping them efficiently. The rearranged and grouped data bits are efficient for lossless compression since it contains less disorder. We justified our claim by taking suitable examples, which clearly shows the benefit of rearranging and grouping of data bits. Note that decoding is straight forward, while sending compressed data we need to send type of rearrangement, best value of k, m and n. The decoder will arrange the data based on this information and get the original data. As a future work it will be interesting to check how this technique work on medical images and X-ray images, since medical records like patient file, X-rays, MRI scan report etc. need lossless compression techniques. It would also be nice to see the loss less JPEG's performance on the given data bits, rearranged and grouped data bits. It is obvious to note that our algorithm can be implemented in parallel, so it would be nice to see the performance of rearranging and grouping on big data.

## References

[1] Ze-Nian Li and Mark S. Drew 2004 *Fundamentals of Multimedia*, Pearson Prentice Hall.

[2] D.A. Huffman 1952 "A Methodfor the Construction ofMinimum-Redundancy Codes' *Proceedings Of the IRE* [Institute of Radio Engineers now the IEEE] 40(9): 1098-1101.

[3] J. Ziv and A. Lempel 1977 "A Universal Algorithm for Sequential Data Compression" *IEEE Transactions on Infonnation Theory* 23(3): 337-343.

[4] J. Ziv and A. Lempel 1978 "Compression ofIndividual Sequences Via Variable-Rate Coding" *IEEE Transactions on Infonnation Theory* 24(5): 530-536.

[5] Rissanen and G.G. 1979 Langdon "Arithmetic Coding," *IB1'vi Journal of Research and Development 23(2):* 149-162.

[6] I.H. Witten, RM. Neal, and J.G. Cleary 1987 "Arithmetic Coding for Data Compression" *Communication of the ACM,* 30(6): 520-540.

[7] M. Kafashan, H. Hosseini, S. Beygiharchegani, P. Pad, and F. Marvasti 2010 "New rectangular partitioning methods for lossless binary image compression," IEEE Int. Conf. Signal Processing, pp. 694–697.

[8] C.E. Shannon 1948 "A Mathematical Theory of Communication' *Bell System Technical Journal* 27: 379-423 and 623-656.

[9] C.E. Shannon and W. Weaver 1949 *The Mathematical Theory oj Communication* Champaign llUniversity of Illinois Press.