Matrix compression methods

Author1[1], Author2[2], Author3[3,*]
1 Author1 Dept/Program/Center, Institution Name, City, State, Country
2 Author2 Dept/Program/Center, Institution Name, City, State, Country
3 Author3 Dept/Program/Center, Institution Name, City, State, Country
* **E-mail: Corresponding author@institute.edu**

Abstract

Author Summary

Introduction

The term compression can be defined as the converting process a data file in another file so that the compressed file size is less than the original [1]. In many cases, data compression is required so that transmission process by internet can occur, even filing and information processing contained in the data file. Although it may be necessary to compress this file can be allocated on a device, for example, data allocation in a computer memory for processing.

Usually the techniques that involve data processing try to work around problems, arise in this case the the data file size, to perform processing. At present and due to the increased speed with which information is generated, it becomes necessary to develop methods which allow such management information in large scale, mostly so that no loss occurs. The information storage on a large scale, especially in digital form, can assist in advancing science [2].

There are different methods that perform this compression procedure, which can be classified based on their characteristics. We highlight the main information loss as characteristics during the compression process, the data manipulation procedure, methodology used, among others [3].

If after compression, data suffer a information loss, this causes loss if the file is uncompressed. The compressed data can not be recovered in its original form. Therefore you can choose the method based on the ratio of the loss characteristics. The data that can be recovered in spite of presenting information loss do not invalidate its use. However, there are other processes that the data must be recovered in full.

According to the loss or not, the information manipulated, one can classify the data types on which compression can occur. When there is information loss, so that this does not undermine the use of the data, we have a method called nondestructive. An example of the data manipulated in this method are sound files (mp3), picture (JPEG) and movies (mpeg). As for the other data after being compressed, it can not have losses, they are called non-destructive. The types of manipulated file by this method are based on text and numbers [1,3].

There is another method classification compression with respect to how it is done, its can be symmetrical or asymmetrical. In the case of symmetrical, time compression and decompression of data occurs synchronously, ie, it take the same time for being compressed and decompressed. In this classification, the fit into based dictionary-algorithm, such as LZW, which use dictionaries for performing the compression. Already another classification, the asymmetric demand different times in implementing the compression and decompression processes, as an example it can highlight the compressing audio method, mpeg [1,3].

Another feature that is considered to distinguish compression types is related to the way in which the process runs. In this new approach, there are methods that are considered adaptive and non-adaptive. As the compression process is executed, the method can be adapted to optimize the processing. Considering this new feature, we can highlight the Golomb code among and other as non-adaptive. Already on the other hand, we have dictionary-based algorithms, LZ77 and LZ78, and adaptive as those based on Huffman coding, arithmetic coding, and others, can be both adaptive and non-adaptive [1,3].

Besides the problem involved with compression, there is another related to the way in which data large volumes must be represented in memory. The procedures described up to now, in large part only function to compress the data file and in some cases access them. When these data should be accessed for manipulation, there is a need to be represented, in the computer case, in memory. From there arise a problem with the representation of this large volume of data in memory. Among the many techniques developed to solve this problem, there is the technique of memory-mapped files that seeks to use main memory to reference the data file in disk as the data are requested, they are loaded into main memory. This technique allows data large volumes can be handled, but still there is a limit, in which case the memory size. Another problem is related to access time data to disk, which is relatively higher compared to the memory access. Some programming languages like Python, R and other, allow the use of this feature. In Python this feature is available in the standard library, already in the R this feature can be used by the package bigmemmory [4].

In this paper we present two methods that allow the compression of large data sets, such as the type of big data, for direct manipulation of files into memory. These methods allow direct access to data in compressed form, its handling and recovery occurs without loss. The proposed method also allows to use the techniques already presented to perform a subsequent compression process only if these data should be stored on physical media. We describe the process of compression and decompression, using as an example strips of bits to represent numerical arrays, the method sparse string [1, 3], to illustrate the methods. We also present a discussion of some results on the characteristics of each proposed method and elucidated the best options in which each can be applied. We have also emphasized that the methods allow information to be accessed after the compression process, which allows the handling and updating, if necessary. At the end of the paper we present their conclusions about using of the proposed methods.

Results

Consider a matrix such that the elements are integers. The results will be presented that indicate the compression process propose two methods. These methods create a new matrix, so that the values of the original matrix are stored in a new matrix, only using less computational resources, in this case memory to be represented.

To check the efficiency of these methods, we consider that this matrix is composed of numbers generated from random variables that follow a probability distributions mixture.

Now for simplicity and functionality will be used by a probability distributions mixture Beta. The probability density function (pdf) of the mixture of two Beta distributions is shown in equation 1. The shape parameters α and β will be correlated to the bit number to represent the values in the data matrix to be compressed. Already the parameter λ defines the proportion of each distribution make mixture. From this value range, one can by distributing generate other and different sets of values to fill the matrix, then one can sweep all possibilities and estimating the efficiency of the compression process.

$$f(x, \alpha_1, \beta_1, \alpha_2, \beta_2) = \lambda \frac{\Gamma(\alpha_1 + \beta_1)}{\Gamma(\alpha_1)\Gamma(\beta_1)} x^{\alpha_1 - 1}(1-x)^{\beta_1 - 1} + (1-\lambda)\frac{\Gamma(\alpha_2 + \beta_2)}{\Gamma(\alpha_2)\Gamma(\beta_2)} x^{\alpha_2 - 1}(1-x)^{\beta_2 - 1} \quad (1)$$

Assume $X_1 \sim Beta(\alpha_1, \beta_1)$ and $X_2 \sim Beta(\alpha_2, \beta_2)$, such that X is mixture resulting of values X_1 and X_2, then X has the probability density function described in Equation 1. Therefore, their mean and variance for a variable X that follows a Beta distribution mixture are shown in Equations 2 and 3.

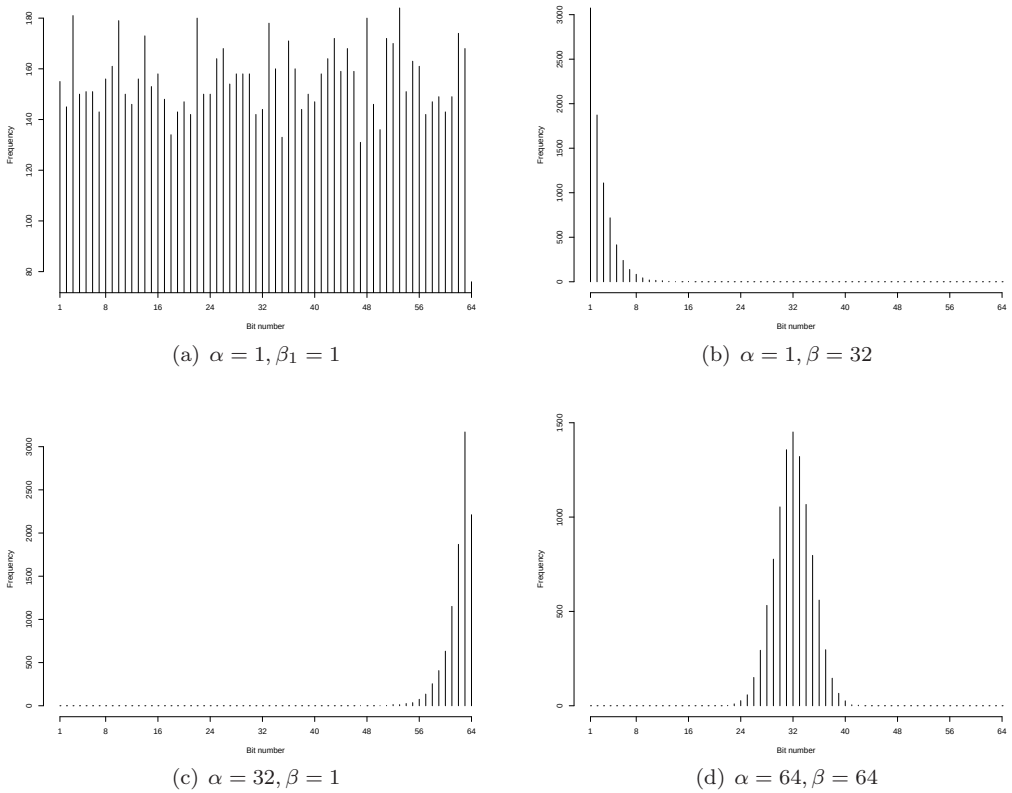
$$E(X) = \lambda E(X_1) + (1-\lambda)E(X_2) = \lambda \frac{\alpha_1}{\alpha_1 + \beta_1} + (1-\lambda)\frac{\alpha_2}{\alpha_2 + \beta_2} \quad (2)$$

$$Var(X) = \lambda Var(X_1) + (1-\lambda)Var(X_2) + \lambda(1-\lambda)(E(X_1)^2 - E(X_2)^2) \quad (3)$$

From these values, which represent the expected value with respect to the parameters α and β, holds up a conversion for representing the bit number required to represent the values in matrix to be compacted. With this, it becomes possible to describe the efficiency as a function of the parameters α and β and the corresponding expectation and variance values generated from this parameter. For a sample of size n, such that the sample fills a matrix with M with r rows and c columns ($n = r \times c$), the expected value for the matrix M can be calculated by the equation 2. From equation 4, we can calculate the expected value for different values of α and β by performing a transformation to adjust the values generated from the mixture of Betas distributions belong to the interval $[1, 64]$. Recalling that the values generated are truncated and only the integer values will be used.

$$\eta(\alpha, \beta) \approx 64 \times E(X) + 1 \tag{4}$$

Defined method for generation of samples to be analyzed, it now remains to estimate the expected efficiency for different values of α and β. To do this, we use the equation 14 and 30, regarding the efficiency of proposed methods 1 and 2. At first, we assume that $\lambda = 0$, ie, the M values matrix are only generated from a Beta distribution. In Figure 1 you can check the different distributions that can be generated from the Beta distribution, only changing the parameters α_1 and β_1 . The histograms represented the distribution of the bit numbers used to represent a sample with 10,000 elements of an array.



(a) $\alpha = 1, \beta_1 = 1$

(b) $\alpha = 1, \beta = 32$

(c) $\alpha = 32, \beta = 1$

(d) $\alpha = 64, \beta = 64$

Figure 1. Histograms constructed from samples with 10,000 elements, generated from the Beta distribution. Below each histogram is possible to verify the parameters used.

As we intend to estimate the efficiency, the equations should be redefined in the sample function as can be seen in Equations 5 and 6, methods 1 and 2, respectively, and $E(B)$ the expected value for the bit number (B) of number sample and k largest expected value this sample.

$$E(\zeta_1) = 1 - \frac{k}{64} \tag{5}$$

$$E(\zeta_2) = 1 - \frac{E(B)}{64} - \frac{k}{64} \tag{6}$$

To define, based on a sample of the matrix M, which method should be used, we calculate the difference $R = E(\zeta_1) - E(\zeta_2)$. When $R > 0$, the first method is more efficient than the method in the second data compression. At market $R < 0$, the second method is more efficient. And otherwise, the two methods are the same.

In this first analysis, it is assumed that the values of α and β belong strictly to the interval $[1, 64]$, the data are generated by modified Beta distribution $Beta(\alpha, \beta)$ and its representing the bit values required to represent elements of a matrix M. Generates three samples with size 10,000, it estimated efficiency values through the equations 5 and 6 and it calculated the difference R.

In Figure 3, you can verify the efficiency of the first (Figure (b)) and second (Figure (c)) methods, the relationship between them (Figure (a)) and the regions defined by the parameters alpha and beta, in which method is more efficient than the other (Figure (d)). In the Figures (b) and (c), it can observe that both methods are effective in the process of data compression, it reaching over 80 % compression. Note that as the average (color scale) of the values generated decreases, the compression rate increase, because the smaller the average value, the smaller bit number needed to represent these values. In the Figure (a) it is possible to verify the relationship between the methods defined by the difference R. Notice that most of the values of the surface defined by R are negative, ie, the second method is more efficient than the first for most combinations. However, there is a small region, which can be seen in Figure (d), color white, indicating that for some values generated from a modified beta distribution with parameters α and β, the first method is more efficient. This region in Figure (A) has red color and it indicates the average of data generated, then the bit number to represent a value in this region is larger and as a consequence first method makes more feasible.

Another analysis can be performed from the combination of two variables, B_1 and B_2, following a Beta distribution with parameters $Beta(\alpha_1,\beta_1)$ and $Beta(\alpha_2,\beta_2)$, respectively, it can calculate the bit number (B) to the mixture of these two distributions. To simplify operations, it is considered that $\lambda = 0.5$, ie, the mixture will have half the numbers generated from the distribution $Beta(\alpha_1, \beta_1)$ and the other part distribution $Beta(\alpha_1, \beta_1)$. The expected value for this mixture is shown in Equation 7.

$$E(B) = 0.5E(B_1) + 0.5E(B_2) \tag{7}$$

Now, we must redefine the functions for calculating the efficiency. Some modifications should be considered, the first being related to Equation 5 which represents the expected value for efficiency using the first method. Thus, the function is defined only in terms of B, since for this method, the important is only the bit greater number than a demand matrix element M (Equation 8). As for the second method, emphasizing that the constant k, is a function of B, again because they are the highest values for represent an element of M (Equation 9).

$$E(\zeta_1) = 1 - \frac{k(E(B))}{64} \tag{8}$$

$$E(\zeta_2) = 1 - 0.5\frac{E(B_1)}{64} - 0.5\frac{E(B_2)}{64} - \frac{k(E(B))}{64} \tag{9}$$

How the values of the alphas and betas strictly belong to the interval $[1, 64]$, ie, representing the values of bits required to represent elements of a matrix M, generating a sample with size $n = 10,000$ and estimated efficiency values through the equations 8 and 9, and soon after, we calculate the efficiency difference R of values calculated.

Before you can calculate the difference of efficiencies, we will analyze some data histograms, constructed from samples with 10,000 elements, generated from beta mixtures distributions. For this, we consider that the values of α_1, β_1, α_2 and β_2 described in Figures 3 and 4. One can see that with different values for the alphas and betas, it becomes possible to construct different histograms. Each histogram of this regards the bit distribution needed to fill a matrix. For each of the samples generated from alphas and betas, their efficiency is calculated methods I and II. The comparison between the methods can be seen in Table 1. The results indicate that over 90 % of the samples with method II was more efficient in the data compression process. This result indicates that for matrices, in which mixtures can be associated probability distributions method II is the most indicated. Analyzing the efficiencies histogram, Figure 5 is verified that the method II (Figure **??**, on average, presents greater efficiency values in the data compression process of the method I (Figure **??**). Again confirming the results presented. Analyzing Figure 6, it can be seen that the regions but which the mean values of mixed distributions, represented by μ_1 and μ_2, indicate that for high values, the method I is more appropriate (Figure **??**), provided that the variability of the data is small (Fig. **??**).

Table 1. Efficiency comparison in the compression data process methods I and II for sample data generated from a mixture of beta distributions parameterized alphas and betas. It was enumerated methods in which each method was more efficient.

Methods	Sample	Percentage
I	592	0.9034%
II	64944	99.0966%
Total	65536	100%

Until now, the method 2 is more effective in the data compression process. This occurs, because if there is some variance in the data set matrix M, the second method can reduce the unnecessary bits loss to represent some numbers, promoting the bit economy. There are cases where the variance between the numbers is null or insignificant. In such cases, the first method is more efficient, since for the same bit number can represent different numbers, for example, 3 bits can represent the values 0 through 7. Therefore, if one matrix is constituted by values between 0 and 7, method I would be more efficient to represent the numbers by a smaller bit number. To simulate the situation described above and the other with the same characteristics a simulation was carried out, generating 10,000 values for a matrix M, so that these values demanding the same amount of bits to be represented. The results for this simulation are shown in Figures 7(a) and 7(b). In Figure 7(a) can be verified that the first method is more efficient in the compression process of the matrix M values for different bit numbers since there are no differences between the bit numbers required to represent each element value of the matrix M. Since they have the same bit number, the first method becomes more effective in this case. In Figure 7(b), we can observe the behavior of the difference in efficiency between the two methods depending on the bit number. In this picture you can see that the measured values are represented by more bits, method 1 is more efficient, but when the values are close to 64-bit, decrease sharply. This decrease occurs when the bit number reaches the value 64, neither method is effective to represent the values, and so there is no compression of the data. It is noteworthy that in this experiment, the method II was no more effective in any situation than method I.

The simulation performed previously, occurs in very special cases, but even so, the method I can compress values generated. Now, let us consider the behavior of the compression process performed on samples generated from the mixing of two dataset. The values generated for each set of data, which

together total 10,000 values, forming the matrix M. All values are represented by two bit quantity and is defined as B_1 and B_2 to methods I and II, respectively. The difference now is that there is a λ, which defines the element number in each group to form the matrix M (Equao 10).

$$n = \lambda b_1 + (1 - \lambda) b_2 \tag{10}$$

The results can be verified in Figure **??**. In Figures 8(a), 8(c) and 8(e), it can be seen surfaces that describe the behavior of the efficiency of method I and II for the values of $\lambda = 0.1$, 0.5 and 0.9, respectively. In these figures, it can be seen that there is a constant behavior for method I, whose behavior is increasing efficiency to values close or equal to B_1 and B_2, while for other values, where there is a difference between the bit number, the efficiency decrease. The method II, different of the method I, has higher efficiency values in regions where the number of bits B_1 and B_2 are discrepant. The difference in the efficiencies of the methods can be verified in Figures 8(b), 8(d) and 8(f). Note that the curve takes positive values for the difference, so the method is the best one. A justification for this behavior is that the bit numbers B_1 and B_2 are close. But for the cases, where the values of B_1 and B_2 are discrepant, method II is the most efficient.

The two methods presented allow the storage and information manipulation in the compressed form, which provides an data managing large amounts optimization. The next step in the work development is the implementation of the proposed methods, as well as tests to be performed on data large volumes.

Discussion

Which method to use?

To know the best methodology to be applied to compress the data matrix, it becomes necessary to evaluate the numbers present in the matrix and as a consequence verify the bits number behavior needed to represent them. The first methodology provides greater efficiency when the bit number to store the information in the matrix are close. The second approach is to try to minimize losses when the numbers have outliers. For example, when it is desired to store values of different magnitudes, such as 0, 1 and 1023. In this values range, it can be seen that the values 0 and 1 only need 1 bit, while the number 1,023 demand 10 bits. Following the first methodology, we would have to divide the strip into regions of 10 bits, which for storing the values 0 and 1, causes loss occurs 9 bits (Equation 11), not allowing for more efficient compression. For work around this problem the second methodology can be applied, promoting data compression. In this case, instead of using 10 bits would be used only 5, with 1 bit to store the value and the other 4 to store the size of the region that this amount will be allocated (Equation 12). See that the second method requires only 24 bits to represent the three values, unlike the first, which requires 30 bits.

$$0000000000000000000000000000000000\underbrace{1111111111}_{1023}\overbrace{\underbrace{0000000000}_{1}}^{\text{Lost}}\underbrace{1}\overbrace{\underbrace{0000000000}_{0}}^{\text{Lost}}\underbrace{0} \tag{11}$$

$$0000000000000000000000000000000000\underbrace{1111111111}_{1023}\underbrace{1010}_{10}\underbrace{1}_{1}\underbrace{0001}_{1}\underbrace{0}_{0}\underbrace{0001}_{1} \tag{12}$$

To verify the efficiency behavior (ζ) compression process data, an analysis will be conducted on the bit number behavior used to store the information in a matrix with r rows and c columns. Consider the following equation 13. This equation gives a percentage of the level of data compression.

$$\zeta = \frac{\text{bits reserved } - \text{bits really needed}}{\text{bits reserved}} \tag{13}$$

Consider a matrix $M_{r\times c}$, with r rows and c columns, we have in total rc elements. To represent these elements in the worst case the, we have to reserve $64 \times rc$ bits. However, if these elements demanding only b bits each represent, in total, would require $b \times rc$ bits. The function efficiency is presented in Equation 14.

$$\zeta = \frac{64 \times rc - b \times rc}{64 \times rc} \tag{14}$$

Simplifying the Equation 14, we obtain the equation 15. The equation resulting from the simplification process does not depend on element number, but the bit number that is used.

$$\zeta(b) = \frac{64 - b}{64} \tag{15}$$

The equation 15 supplies in a direct way to efficiency in the worst and best case compression. In the worst case, $b = 64$, ie, at least one of the elements of the matrix M require 64 bits to be represented, resulting in $\zeta(64) = 0\%$. As for the best case, $b = 1$, the greater efficiency that can be achieved is $\zeta(1) = 98.43\%$. For intermediate cases, it can be seen in Figure 9 efficiency behavior according to the bit number needed (b). This efficiency analysis with respect to the first compression method presented in which all elements require the same bit number to be represented in memory.

For the second method, wherein the compression depends on the value to be stored, it should perform another analysis. This method considers that the elements demanding different bit number and this difference will determine the bit economy. We will analyze this configuration considering some numbers groups that can be represented by the same bit number as possible and when compared with the first method.

Again consider the matrix $M_{r\times c}$, only the rc elements are divided into g groups, where each group has f_i numbers. For each element b_i bits are reserved , corresponding to bits required to represent the desired number and k bits, which represent the bit number to be reserved for storing the strip b_i bits. The efficiency function has to be modified as shown in Equation .

$$\zeta = \frac{64 \times rc - \sum_{i=1}^{g}(b_i + k) \times f_i}{64 \times rc} \tag{16}$$

Equations 17 to 19 are realized manipulations to adjust the equation. The equation that will provide the compression percentage is apresented in Equation 22.

$$\zeta = \frac{64 \times rc - \sum_{i=1}^{g}(b_i \times f_i + k \times f_i)}{64 \times rc} \tag{17}$$

$$\zeta = \frac{64 \times rc - \sum_{i=1}^{g} b_i \times f_i - \sum_{i=1}^{g} k \times f_i}{64 \times rc} \tag{18}$$

$$\zeta = \frac{64 \times rc - \sum_{i=1}^{g} b_i \times f_i - k \times \sum_{i=1}^{g} f_i}{64 \times rc} \tag{19}$$

as

$$\sum_{i=1}^{g} f_i = rc \tag{20}$$

$$\zeta = \frac{64 \times rc - \sum_{i=1}^{g} b_i \times f_i - k \times rc}{64 \times rc} \tag{21}$$

$$\zeta = 1 - \frac{\sum_{i=1}^{g} b_i \times f_i}{64 \times rc} - \frac{k}{64} \tag{22}$$

whereupon

$$k = \pi(\text{ greater value of } M_{r\times c}) \tag{23}$$

In a first example, we suppose again with the matrix M with rc elements, such that the half of elements demanding demand 1 bits to represented and the other half 64 bits to be represented. So in this case i = 2, $b_1 = 1$bits, $b_2 = 64$ bits and $f_i = \frac{rc}{2}$ for $i = 1$ and 2. As the largest demand 64-bit value, then $k = 7$, applying the Equation 23. Efficiency in the bit economy can be calculated using Equation 22, after performing the operations, equations 24 to 28, we obtain the approximate value of 38.29% for compression efficient.

$$\zeta = 1 - \frac{\sum_{i=1}^{2} b_i \times f_i}{64 \times rc} - \frac{7}{64} \tag{24}$$

$$\zeta = 1 - \frac{1 \times \frac{rc}{2} + 64 \times \frac{rc}{2}}{64 \times rc} - \frac{7}{64} \tag{25}$$

$$\zeta = 1 - \frac{32.5 \times rc}{64 \times rc} - \frac{7}{64} \tag{26}$$

$$\zeta \approx 1 - 0.5078 - 0.1093 \tag{27}$$

$$\zeta \approx 38.29\% \tag{28}$$

See that the economy efficiency in using the second method was approximately 38%. What is will influence the size of each number groups, which can be considered as the frequency with which these numbers appear. In the problem presented earlier, we considered two number groups, in which each group had half of elements to be compressed. Let's change the proportion of the groups, but now we use the classical definition of probability to assist in the analysis.

According to the definition the Laplace to the probability refers to the relationship between the case numbers and probable possible case numbers, for example, the relationship between the elements of a subset A, probable case number, with the elements of set Ω , the possible case number, such that $A \in \Omega$. Thus, the probability of the subset A is defined as:

$$P(A) = \frac{\text{Probable case numbers}}{\text{Possible case numbers}} = \frac{\text{Element number in A}}{\text{Element number in } \Omega} \tag{29}$$

With this definition it is possible to associate the problem studied. The sample Ω space is formed by the elements of the matrix to be compacted. The subsets are formed by number groups in the array. Therefore, following this reasoning, we can rewrite the equation 22, being defined in Equation 30. In Equation 30, the term $\frac{f_i}{rc}$ is replaced by p_i, and represents the probability of elements from group i forming the matrix M.

$$\zeta = 1 - \frac{\sum_{i=1}^{g} b_i \times p_i}{64} - \frac{k}{64} \tag{30}$$

with

$$p_i = \frac{f_i}{rc} \tag{31}$$

With the equation 30 can analyze the group proportion influence in the compression efficiency. Consider again the matrix M with r rows and c columns, totaling rc elements. Now consider two groups of numbers whose ratio is defined by Equation 32.

$$p_1 + p_2 = 1,\ p_1 \geq 0 \text{ and } p_2 \geq 0 \tag{32}$$

Thus, p_1 and p_2 define the number proportion of the matrix M requiring 1 bit and 64 bits to be represented, respectively. Thus, efficiency is defined in Equation 33.

$$\zeta = 1 - \frac{1 \times p_1 + 64 \times p_2}{64} - \frac{7}{64} \tag{33}$$

Now, we can verify what the proportions allowing a higher and lower compression from the equation 33. If $\zeta = 1$, then the efficiency is maximum and if $\zeta = 0$, a efficiency is minimum. To calculate the values of p_1 and p_2 so that efficiency is maximum, we must solve the system shown in Equation 34. This system appears from the combination of equations 30, with $\zeta = 0$, with two number sets, one representing the elements represented by 1 bit and other 64-bit, with probability of p_1 and p_2, respectively with the equation 32.

$$\begin{cases} p_1 + p_2 = 1 \\ p_1 + 64p_2 = 7 \end{cases} \tag{34}$$

Solving the system, we have that when $p_1 = 0.9047$ and $p_2 = 0.0953$ efficiency is maximum. Therefore, when approximately 90% of the numbers require only one bit to be represented and approximately 10% require 64 bits, the efficiency is maximum in this case equal to 87.5%. Already in case the efficiency is minimal, we must solve the system shown in Equation 35.

$$\begin{cases} p_1 + p_2 = 1 \\ p_1 + 64p_2 = 57 \end{cases} \tag{35}$$

Thus, when $p_1 = 0.1111$ and $p_2 = 0.8889$ is efficiency minimal according to the second method. For other combinations verify to Table 2. Evaluating the table, you can verify two proportions negative when (p_1,p_2) assume the values (0,1) and (0.1,0.9). It can be interpreted as the need to allocate more memory to use the method, since for the lower efficiency is reached, the value should be $p_1 = 0.9047$ e $p_2 = 0.0953$. In Table, we have an efficiency of 8% in the case of data compression, following the second proposed method, it should have in matrix 20% of the elements represented with 1 bit and 80% represented by 64 bits. If these proportions were maintained and the method was applied first, the efficiency would be minimal in all cases, because the greatest value is represented by 64 bits, which makes the division of the strip to store other values.

Table 2. Combinations p_1 and p_2 to calculate the efficiency.

p_1	p_2	Efficiency
0.0	1.0	-0.109
0.1	0.9	-0.010
0.2	0.8	0.087
0.3	0.7	0.185
0.4	0.6	0.284
0.5	0.5	0.382
0.6	0.4	0.481
0.7	0.3	0.579
0.8	0.2	0.678
0.9	0.1	0.776
1.0	0.0	0.875

Until the moment, we used only two number groups, just a question being examined is how different combinations of these numbers may influence the effectiveness of the compaction process by using two proposed methods.

As an example, we calculated the efficiency for different number groups with the same element number, or rather the same frequency. Note below the efficiencies calculated:

- 3 number groups that require numbers 1, 32 and 64 bits, efficiency $\zeta = 0.3854$,
- 5 number groups that require numbers 1, 16, 32, 48 and 64 bits, efficiency $\zeta = 0.3875$,
- 8 number groups that require numbers 1, 8, 16, 24, 32, 40, 48, 56 and 64 bits, efficiency $\zeta = 0.3888$

When groups are proportionality, ie, have the same frequency, as consequence the same probability of occur, the efficiency is almost the same, using the second method apresented.

Leveraging the idea frequency, which may be generalized as the probability that a number sets can occur, we use probability distributions, in this case the discrete to determine the element numbers in each groupie the frequency. With this, it becomes possible to evaluate, even in an approximate way the proposed methods efficiency behavior. In the case presented earlier, it can be seen that the distribution was uniform discrete was used to determine the bits number, since the probability of occurrence of each group was the same.

To illustrate the calculation, we use three probability distributions: Discrete Uniform, Binomial and Poisson. It is considered that the bit numbers of the numbers to be allocated to matrix follow these three distributions. The respective parameters, thus, relate to the bit number in the uniform discrete case $U(a = 1, b = 64)$, which states that the number probability is represented by 1, 2, 3, ..., 64 is the same, ie, $\frac{1}{64}$. In the binomial case, it is considered that $B(n, 0.5)$, such that the success numbers n represents the maximum bit number to represent the matrix numbers being equal to 1, 8, 16, 32 and 64, with a probability equal to 0.5 assuming a bit value 0 or 1. In the Poisson distribution, the parameter Poisson ($\lambda = p$) being the bit numbers expected to represent the element matrix. For all the simulations, it is considered that $k = 7$, that is, it is considered that there is a possibility of occur a number that requires 64 bits.

Thus, the efficiency is calculated to be as small as possible, because if the value of k decreases, efficiency increases, as can be seen in Equation 30.

To analyze about this new viewpoint, consider a sample of rc elements, corresponding to the matrix element number $M_{r\times c}$. Therefore, a sample generating rc elements can completely fill the matrix and verify the efficiency of the compression method proposed.

Consider that the numbers composing the matrix suggested requiring a bit numbers which follow a discrete uniform distribution $U(a = 1, b = 64)$. To verify the efficiency, generates a sample with 100 ($M_{10\times10}$), 10,000 ($M_{100\times100}$) and 1,000,000 ($M_{1,000\times1.000}$) numbers . Since this is a simulation, are generated 1,000 samples, for each calculated efficiency and the computed final average. The results can be seen in Table 3. If the numbers follow the distribution, the efficiency of the proposed method is approximately 40%.

Table 3. Results of the simulation process efficiency on matrices whose elements are the bits number determined by a Discrete Uniform distribution $U(a = 1, b = 64)$.

Sample size	Efficiency
100	0.3760938
1,000	0.3795953
10,000	0.3826869

We analyzed the performance another simulation, only now the bit number follows a binomial distribution $B(n, 0.5)$, such that the successes number n represents the maximum bit number to represent the

matrix numbers being equal 1, 8, 16, 32 and 64, with a probability equal to 0.5 assuming a bit value 0 or 1. In this case, we analyze the efficiency behavior as a function of the parameter n, the bit number required, in efficiency of data compression. Results can be seen in Table 4. Note that when smaller the bit number to represent array elements, the efficiency level is high using the second method. However, for the case in which the elements require 64 bits, the efficiency is negative. This result indicates that using the second method, when elements require 64 bit this becomes impractical, because it would be require more memory to allocate the compressed matrix.

Table 4. Results of the simulation process efficiency on matrices whose elements are the bit number determined by a binomial distribution $B(n, 0.5)$, such that the successes number n represents the maximum bit number to represent the matrix numbers being equal to 1, 8, 16, 32 and 64, with a probability equal to 0.5 assuming a bit value 0 or 1.

	Efficiency		
	Sample size		
Strip size	100	1,000	1,000,000
1	0.9760937	0.9764797	0.9765566
8	0.8109375	0.8128453	0.8125079
16	0.626875	0.6251859	0.6249333
32	0.2476562	0.2504953	0.2499712
64	-0.5004688	-0.4999828	-0.5000366

It is now considered that the bit number distribution is described by a Poisson (λ). Now the parameter λ can be considered as the expected bit number to represent numbers in matrix. Values greater than 64 generated in this simulation were considered as being equal to 64. The results for this simulation can be seen in Table 5. Note that increasing the bit number to represent numbers increases, there is a loss of efficiency in the compression process. Another interesting fact is the emergence of a negative efficiency. In this case, for values generated with a $\lambda = 64$, the second method requires more memory to allocate the matrix formed by elements generated from this distribution.

Table 5. Results of the simulation process efficiency on matrices whose elements are the bit number determined by a Poisson distribution (λ), such that λ represents the bit number to represent the expected matrix numbers.

	Efficiency		
	Size sample		
λ	100	1,000	1,000,000
1	0.8746875	0.8749859	0.8749834
8	0.7657812	0.7656016	0.7656138
16	0.6300000	0.6400594	0.6405952
32	0.3851562	0.3903531	0.3903477
64	-0.0528125	-0.05984375	-0.05953145

As the bits amount that form the numbers composing the matrix M are considered random variables, we can perform an analysis of the expected values and the variability of the data to verify the compression efficiency level.

Consider that the bit number (B) to represent the M matrix elements are defined as a random variable, $B \sim U(a = 1, b = 64)$, then applying the expected value in the equation 36, we have

$$E(\zeta) = 1 - \frac{\sum_{i=1}^{g} b_i \times p_i}{64} - \frac{k}{64} \tag{36}$$

as $E(b_i) = \sum_i b_i \times p(b_i) = \frac{a+b}{2}$, it result in

$$E(\zeta) = 1 - \frac{E(B)}{64} - \frac{k}{64} \tag{37}$$

Recovered the example used, consider that there are two groups, $b_1 = 1$ and $b_{64} = 64$, implying that $k = 7$. Since $E(X) = \frac{a+b}{2}$, when $X \sim U(a, b)$, the efficiency is defined in Equation 38.

$$E(\zeta) = 1 - \frac{\frac{1+64}{2}}{64} - \frac{7}{64} \approx 38.29\% \tag{38}$$

This result corroborates with the findings anteriorly found in Table 3. The same principle can be applied when the bit number (B) is a random variable with Binomial and Poisson distributions already displayed. In the case of Binomial when $B \sim B(n = 64, p)$, has

$$E(\zeta) = 1 - \frac{\sum_{i=1}^{g} b_i \times p_i}{64} - \frac{k}{64} \tag{39}$$

in this case $E(b_i) = \sum_i b_i \times p(b_i) = n \times p$

$$E(\zeta) = 1 - \frac{64 \times p}{64} - \frac{k}{64} \tag{40}$$

Again, it is considered that $n = 64$ and $p = 0.5$ as the success probability of occurring a number that requires 64 bits to be represented, then $k = 7$. As a result we obtain

$$E(\zeta) = 1 - \frac{64 \times 0.5}{64} - \frac{7}{64} \approx 39.05\% \tag{41}$$

The value found can be seen in Table 4, obtained through simulations. As for the Poisson case, the efficiency value is defined as

$$E(\zeta) = 1 - \frac{\sum_{i=1}^{g} b_i \times p_i}{64} - \frac{k}{64} \tag{42}$$

in this case $E(b_i) = \lambda$,

$$E(\zeta) = 1 - \frac{\lambda}{64} - \frac{k}{64} \tag{43}$$

We consider $\lambda = 1$,

$$E(\zeta) = 1 - \frac{\lambda}{64} - \frac{k}{64} = 1 - \frac{1}{64} - \frac{7}{64} \approx 87.5\% \tag{44}$$

The efficiency level calculated $\zeta \approx 87.5\%$ can be seen in Table 5. The results presented show simulations and verify that the compression process occurs in different cases.

In all tests so far conducted, we analyzed the impact of the average data compression efficiency. The final analysis to be performed, consider the mean and variance of the numbers to be compressed. Thus, it is expected function and efficiency in calculating the mean and variance of matrix numbers. (Remover?)

Materials and Methods

1 Matrix compression

The matrix compression process consists in reducing the bytes number required to allocate a matrix in memory. This process allows a greater amount of information to be stored using the same hardware available memory allocation traditional method. One of the advantages in applying this method is the possibility to work with large information amounts about data directly from memory without need for disk access, reading and writing, which burdens the processing.

The steps and implementations required to use the prepossessing method are described below. Consider a matrix $M_{r\times c}$ in which r is the rows number and c the columns number. Each element of this matrix, called m_{ij}, only stores an integer number. For an integer number being stored in the memory of a 64-bit processor, using the largest number of available bits, demand is 64 bits. Therefore, the bit number (Δ) required to allocate all the elements of a matrix M in the traditional manner may be defined by:

$$\Delta = l \times c \times 64 \tag{45}$$

The Δ represents a greater bit number than that actually used. Assume that in a matrix, for example, there are only elements with values equal to 0 or 1. These values are represented in binary by only one bit. Therefore, when one is allocated in variable that requires 64 bits to be stored in memory, there is a loss of 63 bits, which could be used to store other numbers or information. This example deals with the case more expressive with respect to waste memory. The bit number calculation δ really necessary to implement this example is represented in the Equation 46.

$$\delta = l \times c \times 1 \tag{46}$$

The bits economy ξ can be represented by the difference between Δ and δ (Equations 47 to 50).

$$\xi = \Delta - \delta \tag{47}$$

$$\xi = r \times c \times 64 - r \times c \times 1 \tag{48}$$

$$\xi = r \times c \times (64 - 1) \tag{49}$$

$$\xi = r \times c \times 63 \tag{50}$$

Based on the Equations 45 to 50, it is can to observe that the economy ξ is high, because $r \times c$ elements require only $r \times c \times 1$ bits instead of $r \times c \times 64$ bits. The same happens with other values that do not need 64 bits to be represented in memory, because it not always all bits are used. It can be said that the economy grows linearly with the number of array elements.

There are two ways to perform the manipulations of these bits to minimize wasted space. Each of these forms follows a specific methodology to be applied to certain cases.

The first method: Analyzing the greatest value to allocate in a matrix

The first methodology consists in determining the value of the largest element θ, which should be stored in the matrix and as a result determines the largest bit number (π) really need to store that value. The bit number (π) can be directly approximated by the equation 51. This approach consists in calculating

the bit value needed to represent the value, as there is usually decimal, the value 1 which is added ensures, that the resulting bit number to be correct to store such value, the integer part of the operation.

$$\pi(\theta) \approx \begin{cases} 1, & \text{if } \theta = 0 \\ \left(\frac{log(\theta)}{log(2)}\right) + 1, & \text{if } \theta > 0 \end{cases} \tag{51}$$

Each matrix element of M, denoted by m_{ij}, is represented by a strip composed of 64 bits. In the example mentioned, for values 0 and 1, the value of θ equals 1, ie, in a strip of 64 bits may be allocated 64 values. Each bit of this strip will represent one of the values to be allocated. This fact is responsible for saving bits, for 64 values, which were traditionally stored in $r \times c \times 64$ bits are stored in only $r \times c$ bits.

Suppose that the greatest value to be stored in a matrix Φ is $\theta = 1023$. Therefore, the bit number required to represent that really value is $\pi = 10$, because the number 1023 in binary base is 1111111111, therefore, demand 10 bit and the other 54 bits zeros disregarding the right, to be represented in memory. Consider a matrix Φ represented in equation 52.

$$\Phi = \begin{bmatrix} 900 & 1023 & 721 & 256 & 1 & 10 & 700 & 20 & \dots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix} \tag{52}$$

The values of the matrix Φ in binary are represented in Table 6. It is possible checking the bit number necessary to represent the values, then it can be concluded that with 10 bits, it is can represent all numbers.

Table 6. Some matrix elements Φ represented in binary base. It is can to verify that using 10 bits is possible to represent the all values.

Element	Value	Binary	Bit number necessary
$\Phi_{1,1}$	900	1110000100	10
$\Phi_{1,2}$	1023	1111111111	10
$\Phi_{1,3}$	721	1011010001	10
$\Phi_{1,4}$	256	100000000	9
$\Phi_{1,5}$	1	1	1
$\Phi_{1,6}$	10	1010	4
$\Phi_{1,7}$	700	1010111100	10
$\Phi_{1,8}$	20	10100	5

Consider a matrix ϕ, such that the matrix will store the the matrix Φ values, only using the bit number actually required so that there is not wastage. According to this methodology, the first element of the matrix ϕ, denoted by $\phi_{1,1}$, which has a strip of 64 bits associated, it will be partitioned into regions. Each of these regions will store the values of the matrix Φ. As it takes 10 bits to represent the values of the matrix Φ, the bit element strip $\phi_{1,1}$ will contain 6 regions, such that each region will bundle 10 bits. So overall, this strip will have 6 regions, totaling 60 bits. The 4 bits free remaining it would be used later. The elements bit strip of $\phi_{1,1}$ can be verified in the equation 53. It is possible to observe the regions, the values stored (binary and decimal), and the bits that are left initially.

$$\phi_{1,1} = 0000\underbrace{0000001010}_{10}\underbrace{0000000001}_{1}\underbrace{0100000000}_{256}\underbrace{1011010001}_{721}\underbrace{1111111111}_{1023}\underbrace{1110000100}_{900} \tag{53}$$

The process of allocation values is performed as follows: the element $\Phi_{1,1} = 900$ is stored in the first region of the element strip $\phi_{1,1}$, which corresponds to bits 0 to 9. Already for the element $\Phi_{1,2} = 1,023$ is allocated in the second region, from 10 to 19 bits and the same with the other elements. Thus, the

elements $\Phi_{1,i}$, with $i = 1, \ldots, 7$ are stored on the strip element $\phi_{1,1}$. Already the seventh element, $\Phi_{1,7} = 700$, should be divided into two parts, so that a part are stored in the strip element $\phi_{1,1}$ and the other in the element strip $\phi_{1,2}$. The process of division shown in equation 54, it can be seen the place where the split occurs strip element $\Phi_{1,7} = 700$ in the parts a and b. In the equation 55, it can verify the bits grouped, demonstrating that it is possible to recover the information.

$$\underbrace{\underbrace{0000010100}_{20}\overbrace{101011}^{a}}_{\phi_{1,2}} \mid \underbrace{\overbrace{1100}^{b}\underbrace{0000001010}_{10}\underbrace{0000000001}_{1}\ldots}_{\phi_{1,1}} \tag{54}$$

$$\underbrace{0000010100}_{20}\underbrace{\overbrace{101011}^{a} \mid \overbrace{1100}^{b}}_{700}\underbrace{0000001010}_{10}\ldots \tag{55}$$

In a general way, what happens is the storage of the matrix elements Φ in a matrix ϕ, such that the bits required to store them in the array ϕ is much smaller than the matrix Φ, due to reorganization of the data. Even with the compression of the data, the information can be retrieved, thus justifying the proposed procedure.

The second method: Analyzing the values that will be allocated in a matrix

A second proposed method tries to make a new form of data compression. Consider again the matrix described in Equation 52. Suppose the largest value stored in it is number 1023. As already demonstrated, to store this value in binary are necessary 10 bits. Now instead of reserving a region on the strip to store each value, reserves two regions. Such regions, one is used to store the bit number needed to represent the value and the second region will store the respective value. The first region will have a fixed size, in the example cited, 4 bits. The 4 bits are required to store the bit number necessary to store the largest value, in this case 10 bits. The size of the second region is determined by the value stored in the first. And so the other values are stored.

To illustrate this methodology, we will be using the matrix represented in Equation 52, called Φ. The largest element of Φ is 1,023, as soon as the bit number required to represent this value is $\pi = 10$ bits. The element $\Phi_{1,1} = 900$ is stored in bits of the element strip of $\phi_{1,1}$. To store the element $\Phi_{1,1}$, it should create the two first regions of the strip element $\phi_{1,1}$. Initially element strip $\phi_{1,1} = 0$ is shown in equation 56.

$$\phi_{1,1} = 0000000000000000000000000000000000000000000000000000000000000000000000000000 \tag{56}$$

The first region that will store the value 10, because to represent the number 900 are need 10 bits. Then, it will have 4 bits, because represent the value 10, it demand 4 bits. The 4 bits are manipulated of according to the bit number to represent the largest value of Φ, in which case, the value is 1,023. The strip $\phi_{1,1}$, after storing the information of $\Phi_{1,1}$ is shown in Equation 57

$$\phi_{1,1} = 0000000000000000000000000000000000000000000000000000\underbrace{\underbrace{1110000100}_{\text{value=900}}\underbrace{1010}_{\text{10 bits}}}_{900} \tag{57}$$

Adding the elements $\Phi_{1,2} = 1023$ and $\Phi_{1,3} = 721$, the strip of $\phi_{1,1}$ updated is represented in Equation 58. You can check the regions that store the information from the elements.

$$\phi_{1,1} = 00000000000000000000\underbrace{1011010001}_{721}\underbrace{1010}_{10}\underbrace{1111111111}_{1023}\underbrace{1010}_{10}\underbrace{1110000100}_{900}\underbrace{1010}_{10} \tag{58}$$

Until this moment, the second method is using more bits to store a value that the first methodology. With the insertion of the fourth element that the bit economy occurs, because $\Phi_{1,4} = 256$ demand only 9 bits. The first region with 4 bits defines how many bits the second region will contain the stored value. So instead of storing $\Phi_{1,4} = 10$ in 10 bits, as in the previous method, it demands 9 bits, but it takes a whole 13 bits. The economy becomes evident when the element $\Phi_{1,5} = 1$ is stored. Note that this element requires only 1 bit to be stored, so instead of 10 bits, only 1 bit is used, more 4 bits to indicate that the value is stored in only 1 bit. Therefore, instead of the 10 bits required to store that value pair following the first methodology, now only 5 bits are needed. The strip of $\phi_{1,1}$, after insertion of the latter two values are updated in Equation 59.

$$\phi_{1,1} = 0000\underbrace{1}_{1}\underbrace{0001}_{1}\underbrace{100000000}_{256}\underbrace{1001}_{9}\underbrace{1011010001}_{721}\underbrace{1010}_{10}\underbrace{1111111111}_{1023}\underbrace{1010}_{10}\underbrace{1110000100}_{900}\underbrace{1010}_{10} \tag{59}$$

Now the element $\Phi_{1,6} = 10$ will be inserted in the strip $\phi_{1,1}$. The process need that 4 bits are used element strip $\phi_{1,1}$ and another 4 bits of the element $\phi_{1,2}$. The strips updated element $\phi_{1,1}$ and $\phi_{1,2}$ can be verify in Equations 60 and 61.

$$\phi_{1,1} = \underbrace{0100}_{4}\underbrace{1}_{1}\underbrace{0001}_{1}\underbrace{100000000}_{256}\underbrace{1001}_{9}\underbrace{1011010001}_{721}\underbrace{1010}_{10}\underbrace{1111111111}_{1023}\underbrace{1010}_{10}\underbrace{1110000100}_{900}\underbrace{1010}_{10} \tag{60}$$

$$\phi_{1,2} = 00000000000000000000000000000000000000000000000000000000000000\underbrace{1010}_{10} \tag{61}$$

Adding the last two elements $\Phi_{1,7} = 700$ and $\Phi_{1,8} = 20$ in the element strip $\phi_{1,2}$, which after being updated, it can be viewed at equation 62.

$$\phi_{1,2} = 00000000000000000000000000000000\underbrace{10100}_{20}\underbrace{0101}_{5}\underbrace{1010111100}_{700}\underbrace{1010}_{10}\underbrace{1010}_{10} \tag{62}$$

For storing the elements of matrix Φ in the matrix ϕ were used 87 bits instead of 80 bits used in the first method. The choice of which method to use will be discussed in the section Discussion. The proposed methods have different levels of compression, depending directly from the values that must be stored.

Acknowledgments

References

1. Salomon D, Motta G, Bryant D (2009) Handbook of Data Compression. London: Springer, 1359 pp.
2. Lynch C (2008) Big data: How do your data grow? Nature 455: 28–29.
3. Salomon D (2007) Data Compression: The Complete Reference. Number v. 10 in Data compression: the complete reference. London: Springer-Verlag New York Incorporated.
4. Kane MJ, Emerson JW (2012) bigmemory: Manage massive matrices with shared memory and memory-mapped files. URL `http://CRAN.R-project.org/package=bigmemory`. R package version 4.3.0.

Figure Legends

Tables

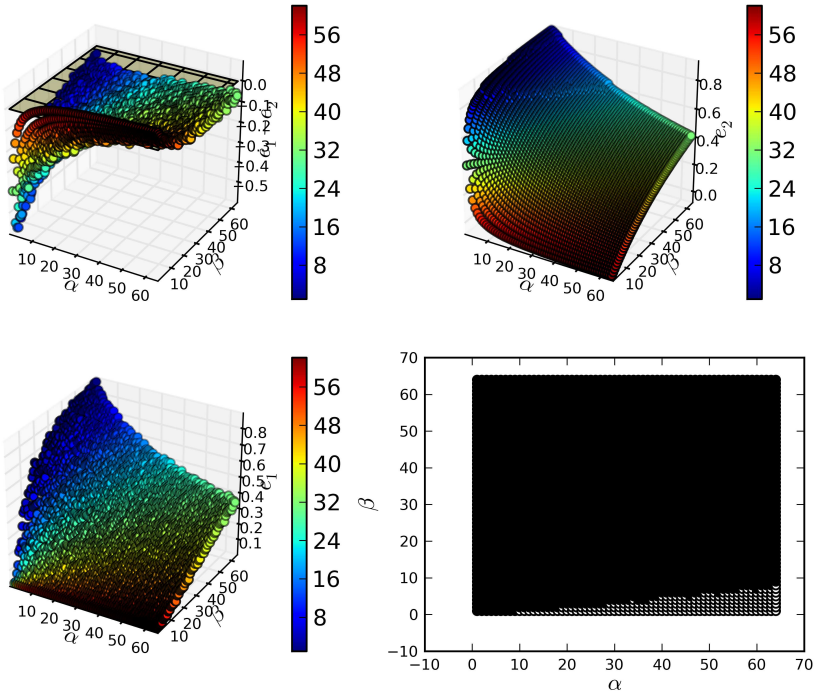
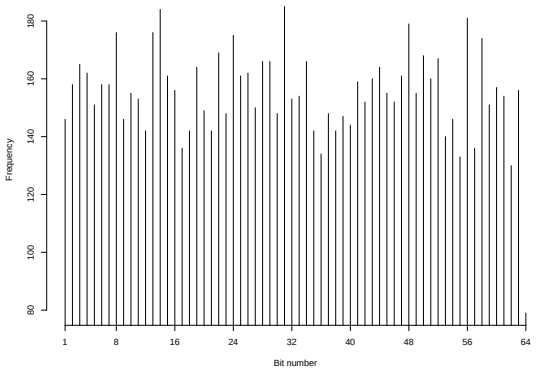
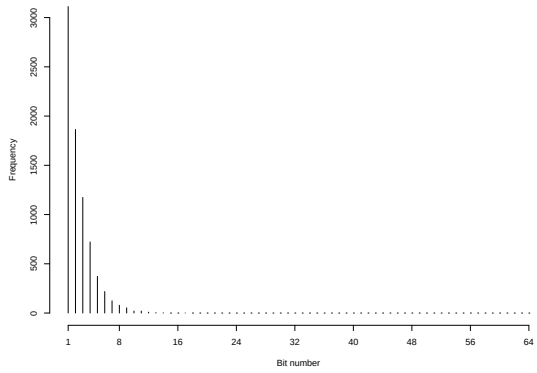


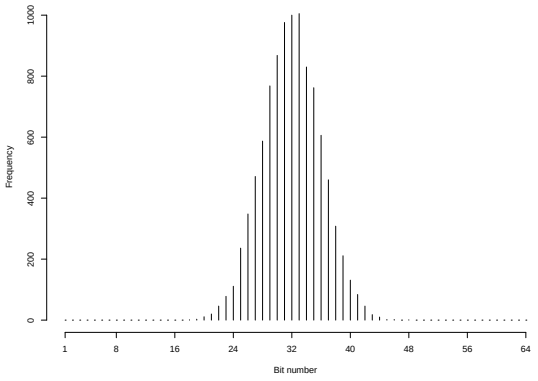
Figure 2. Comparisons between methods I and II in relation to efficiency in the compression process. In the Figure (a) can see the difference between the efficiencies of each method R. It is verified that practically the entire surface has $R < 0$, that is, the second method is more feasible for compressing the information generated from a Beta distribution modified $Beta(\alpha, \beta)$. However there is a region, which can be seen in Figure (d), color white, its indicates the values generated from values determined by α and β, the first method is more efficient. This region can be identified in Figure (a) in red, indicating that the generated values have larger average and require more bits to be represented. This demand makes the first method more feasible for the values generated in this configuration. In Figures (b) and (c), it can be seen that both methods are effective with respect to high data compression rates generated from a Beta distribution $Beta(\alpha, \beta)$. Note, in both figures, the value of the efficiency of the compression increases with reduction of the average number of generated. This is due to reducing the bit number required to represent those values.



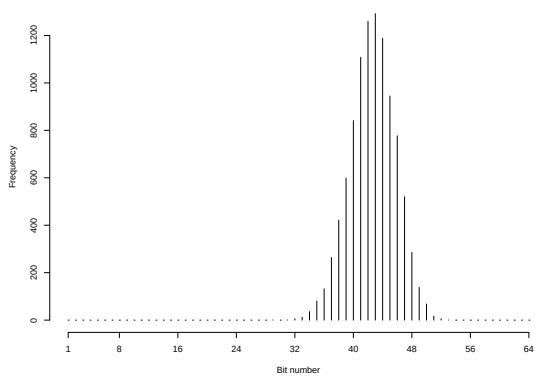
(a) $\alpha_1 = 1, \beta_1 = 1, \alpha_2 = 1, \beta_2 = 1$



(b) $\alpha_1 = 1, \beta_1 = 32, \alpha_2 = 32, \beta_2 = 1$

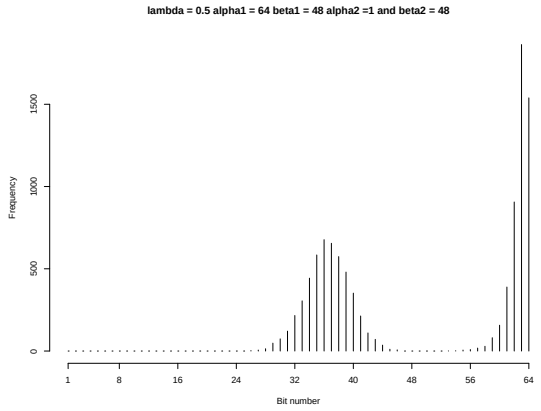


(c) $\alpha_1 = 32, \beta_1 = 32, \alpha_2 = 32, \beta_2 = 32$

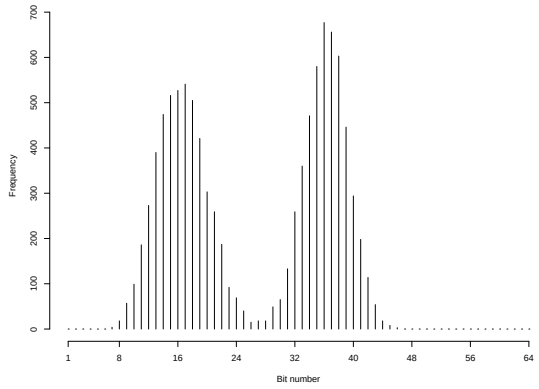


(d) $\alpha_1 = 64, \beta_1 = 32, \alpha_2 = 32, \beta_2 = 64$

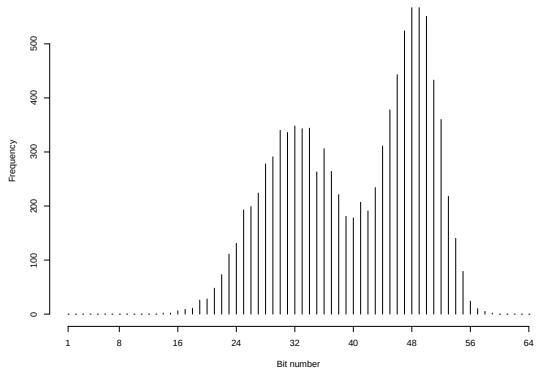
Figure 3. Histograms constructed from samples with 10,000 elements, generated from the combination of two distributions Betas. Below each histogram is possible to verify the parameters used..



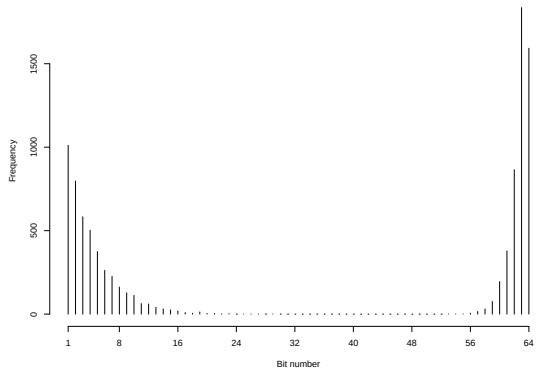
(a) $\alpha_1 = 64, \beta_1 = 48, \alpha_2 = 1, \beta_2 = 48$



(b) $\alpha_1 = 16, \beta_1 = 46, \alpha_2 = 49, \beta_2 = 64$

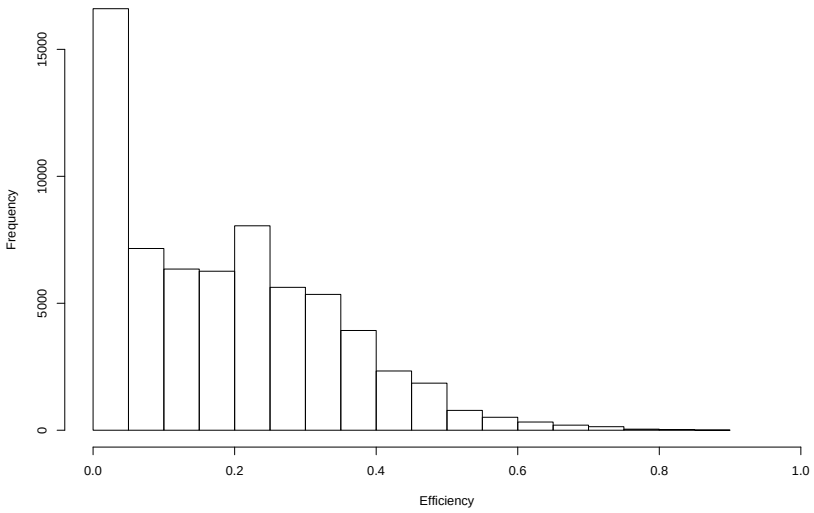


(c) $\alpha_1 = 16, \beta_1 = 16, \alpha_2 = 16, \beta_2 = 49$

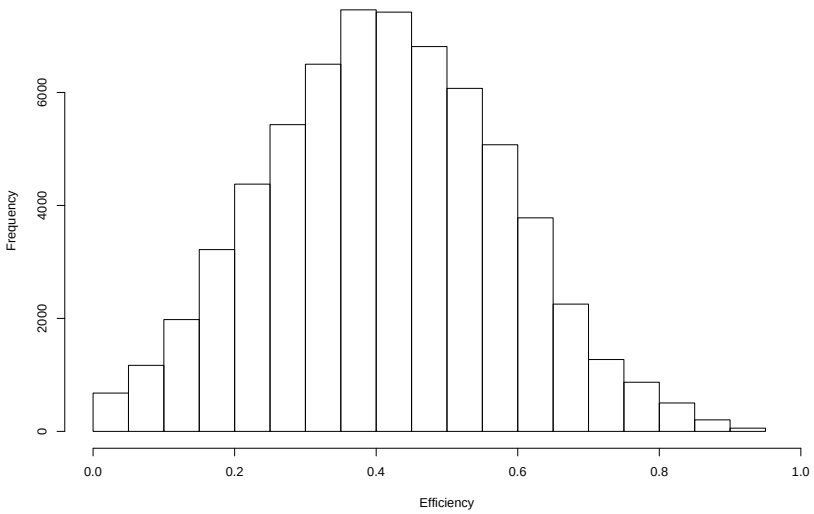


(d) $\alpha_1 = 1, \beta_1 = 16, \alpha_2 = 1, \beta_2 = 49$

Figure 4. Histograms constructed from samples with 10,000 elements, generated from the combination of two distributions Betas. Below each histogram is possible to verify the parameters used.

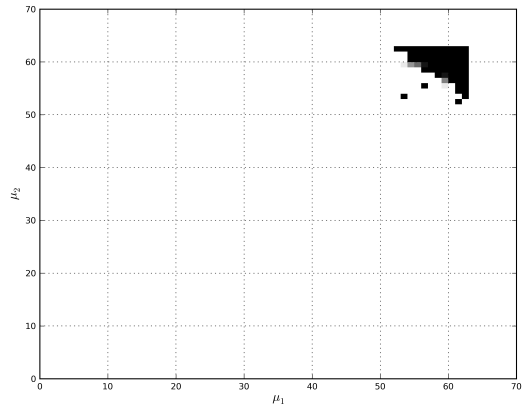


(a) Efficiency histogram of the method I

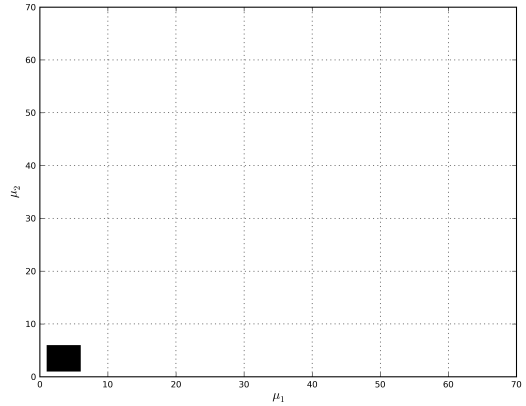


(b) Efficiency histogram of the method II

Figure 5. Efficiency histograms of the methods I and II. It is verified that the method II gives an average efficiency greater than method I.

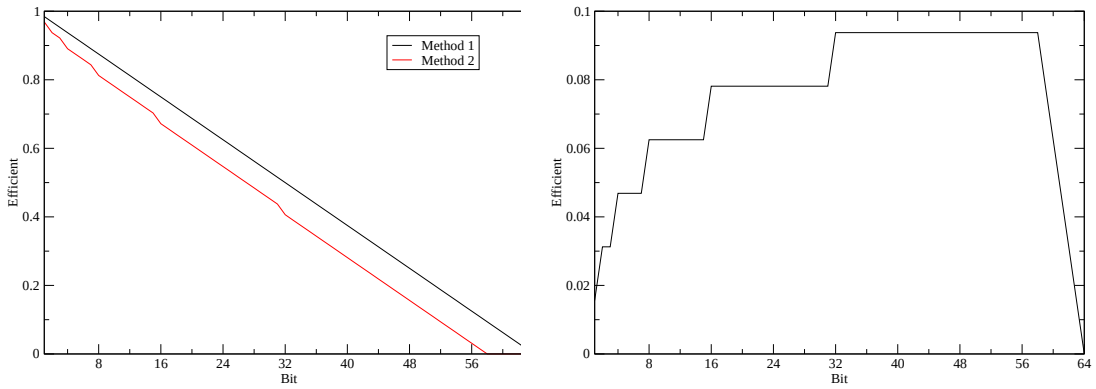


(a) Efficiency of the methods I and II as a function of the mean distributions mixed.



(b) Efficiency of the methods I and II as a function of the variance distributions mixed.

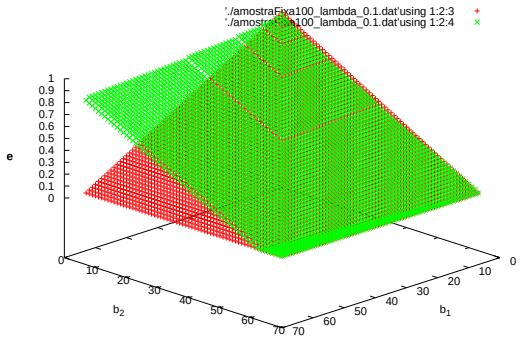
Figure 6. Mean values of the distributions mixed in which method I and II are more efficient. Note that only a small region with high values for the mean distributions mixed that the method I is more efficient, in other regions the method II is highlighted. With respect to variance of distributions mixed, one can verify that the method I is more efficient for low values of the variance.



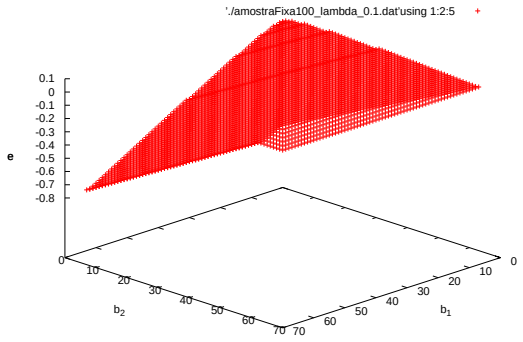
(a) Comparison of the efficiency of the methods I e

(b) Difference in efficiency of the methods I e II.

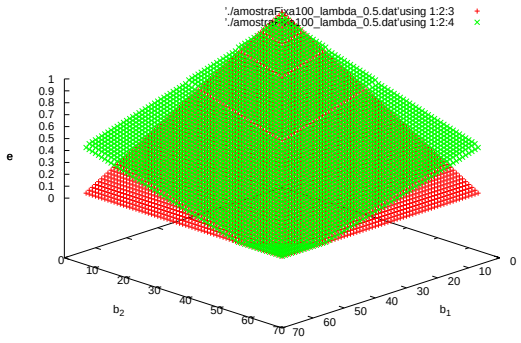
Figure 7. Efficiency of the methods I and II the data compression process of a matrix of numbers which require the same amount of bits to be represented.



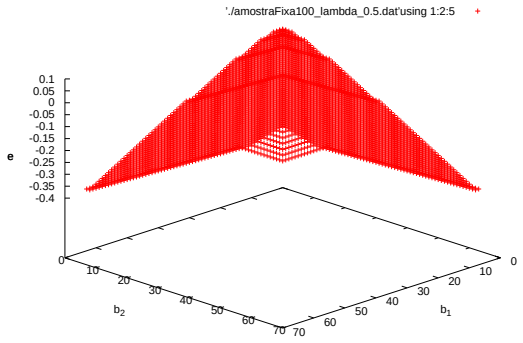
(a) Comparison of the efficiency of the methods I and II.($\lambda = 0.1$)



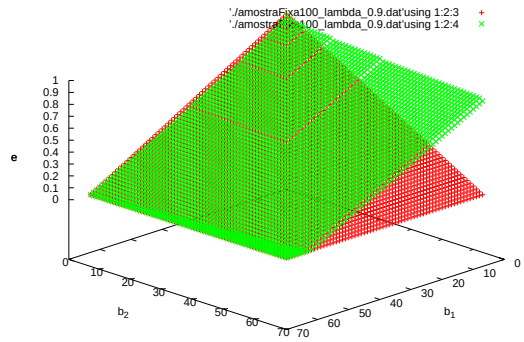
(b) Difference in efficiency of the methods I and II.



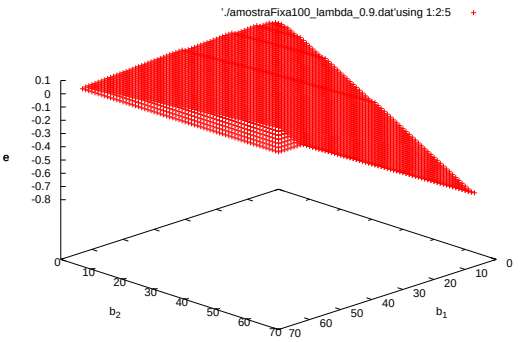
(c) Comparison of the efficiency of the methods I and II.($\lambda = 0.1$)



(d) Difference in efficiency of the methods I and II.



(e) Comparison of the efficiency of the methods I and II.($\lambda = 0.1$)



(f) Difference in efficiency of the methods I and II.

Figure 8. Efficiency of the methods I and II in the data compression process of a matrix of numbers which require the same amount of bits to be represented.

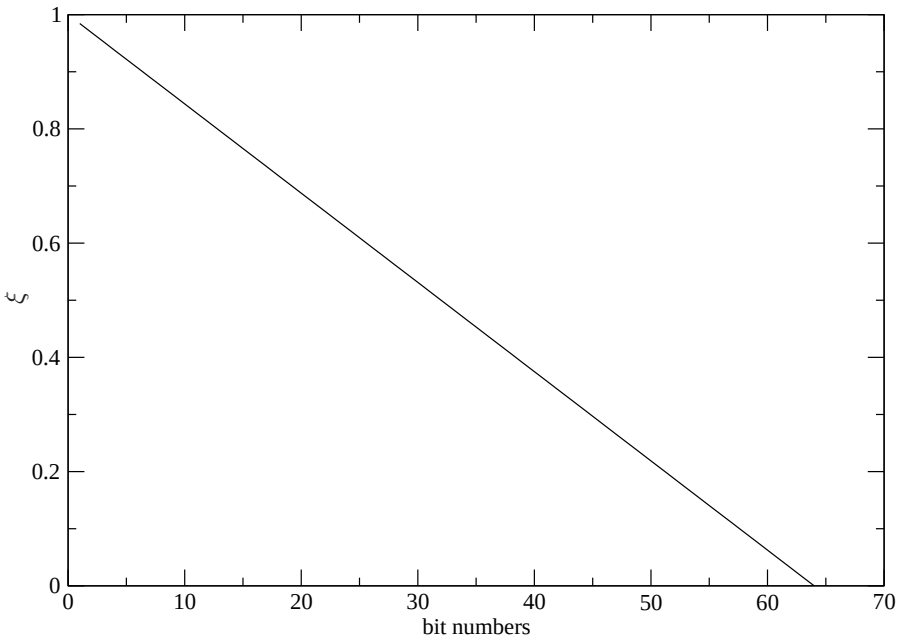


Figure 9. Compression efficiency behavior with respect to bit number.