Journal of Theoretical and Applied Information Technology
15th April 2019. Vol.97. No 7
© 2005 – ongoing JATIT & LLS

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

A NEW PERMUTATION METHOD FOR SEQUENCE OF ORDER 2^8

[1]ABDULLAH AZIZ LAFTA, [2]AMMAR KHALEEL ABDULSADAH, [3]SAFAA JASIM MOSA

[1,2,3] University of Kufa, Faculty of Education for Girls, Department of Computer Science, Najaf, Iraq

E-mail: [1]abdullaa.lafta@uokufa.edu.iq, [2]ammar.khaleel@uokufa.edu.iq, [3]safaaj.aljuburi@uokufa.edu.iq

ABSTRACT

Permutation is reordering for a set of objects with taking into consideration the important of order of their locations. In order to increase security performance, this paper presents a new method of permutation for sequence of order 2^8. In this propose method, any information can be converted to the sequence of approximate values between 0 and 255 as a set of blocks. It is needed to build the so called lookup table so that it contains the frequency and positions of the original sequence values. In the experiments, twenty-six image files have been experimented as evidence to the proposed method. In addition, the correlation and entropy measures are computed to test the quality of permutation. All the tests observed that method have significantly effective in reducing the correlation and thereby decreasing the perceptual information of the sequence. Hence, the security is improved.

Keywords: *Permutation, Algorithm, Frequency, Sequence, Byte*

1. INTRODUCTION

In statistics, permutation is a method to make changes on the elements of samples of data to measure the significant extent of original data. The data sequence can be divided into blocks for analysis purposes [1]. Nowadays, many works in different fields, such as signal processing [2], digital image encryption [3][4][5], bioinformatics [6], neuroimaging [7] and so on use permutation to understand the contents of data to solve a particular problem, such as the highest correlation in data sequence.

The challenge in the permutation field is getting the highest benefits offered from the set of assumptions. Furthermore, a specific type of data adapts with a specific permutation approach. Permutation approaches that are good for textual data may not be suitable for multimedia data [8]. In most of the natural images, the values of the neighboring pixels are strongly correlated. This means that the value of any given pixel can be reasonably predicted from the values of its neighbors [9]. This information is reduced if decreasing the correlation between the image elements using certain permutation process. As a result, when the correlation is decreased, the values of its neighbor's elements become difficult to predict [3].

In the digital data domain, the smallest element of data is the binary bit in which the data element (basic unit of information) is represented using the binary number system of ones (1) and zeros (0). Since the byte is a unit of digital information that consists of eight bits, it is permitting the values from 0 to 255. A combination of bytes is referring to one isolated item of information [10].

Data bytes are formed as a set of a sequence of elements which stored at a specific location, for example in a file to be serialized in a file format. The file can be decomposed into blocks or sequences where assuming that the block size is N elements. Small blocks are preferred by some researchers in computer security [4, 5].

The main idea behind the present work is that any source of data, for example a file, can be viewed as an arrangement of N-blocks. The intangible information that presented in that source is due to the correlations among the N-blocks in a given arrangement. This information can be reduced by decreasing the correlation among the N-blocks using the proposed permutation method. In this study, we try to indexing the contents of a sequence into structure that depends on the frequency of each value and the locations of that value within the sequence. The purpose of this approach is to get a complete different structure to the original while retaining the information between the values if a reverse approach has been applied.

The rest parts of this paper are organized as follows: section two presents the propose methodology in details. Section three shows the experimental results and discussion while the conclusions are summarized in section four.

2. RESEARCH METHODOLOGY

The research methodology is concern with the input sequence of distinct values to build the lookup table for location of these values which help to compute new quantities. These quantities are combined to form the output sequence

2.1. Input Sequence: Definition and Properties

The input for the proposed is a sequence of elements, denoted by S=(V,L). This sequence supposes it contains strong information, which is the orders of its elements, see Figure 1. The sequence has a finite or limited number of elements, say N+1 elements. Here, the maximum possible value could be for N is 255×256+255. As illustrate in figure (1), the two sets V and L are V is the list of all element values appear in S, and L is the list of the corresponding locations of these values where they appear in S.

$V=\{v_0,v_1,\dots,v_N\}$ and $L=\{l_0,l_1\dots,l_N\}$ (1)

Also, the input sequence S can be seen as a sequence of ordered pairs of objects as follows:

$S=\{(v_0,l_0),(v_1,l_1),\dots,(v_N,l_N)\}$ (2)

Define the set B is, $B=\{x: 0 \le x \le 255\}$ contains only non-negative integers that less than 256. Then $v_k \in V$ is must be also $v_k \in B$.

Each element, say e, in the sequence S has two objects: value, $v_e \in V$, and an index or a location, $l_e \in L$, i.e. $e = (v_e, l_e)$. The element values are belong to the set B, i.e. $0 \le v_e \le 255$ for any e such that $0 \le e \le N$. That is, $V= \{v_e: v_e \in B \ \forall\ 0 \le e \le N\}$. Whereas, the locations of elements are belonging to the range $0\cdots N$. i.e. $L= \{l_e: 0 \le l_e \le N \ \forall\ 0 \le e \le N\}$. The element values are explicit information, where element location values are implicit information.

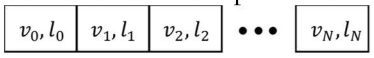


Figure 1: Input Sequence(S)

2.2. Output Sequence: Definition and Properties

The results of the proposed method are four sequences of elements, denoted by QL, RL, QF, and RF. These can be written as follows:

$QL= \{q_0, q_1,\cdots,q_N\}$and $RL=\{r_0,r_1,\cdots,r_N\}$ (3)

$QF=\{q_0,q_1,\cdots,q_{255}\}$ and $RF=\{r_0,r_1,\cdots,r_{255}\}$ (4)

Such that, $q_i \in B \ \forall\ q_i \in QL$ and $r_i \in B \ \forall\ r_i \in RL$ for $0 \le i \le N$. Also, $q_x \in B \ \forall\ q_x \in QF$ and $r_x \in B \ \forall\ r_x \in RF$ for $0 \le x \le 255$.

Where QL and RL are two sequences related to the set L, such that:

$l_i=q_i\times256+r_i$ for $q_i \in QL$ and $r_i \in RL \ \forall\ 0 \le i \le N$ (5)

The other two sequences, QF, and RF are related to frequencies of values in input sequence S. That is, for any value $x \in B$, there is a counter for how many times that value x is repeated in sequence S. Call this value as the frequency of value x, denoted by f_x, such that:

$f_x=q_x\times256+r_x$ where $q_x \in QF$ and $r_x \in RF \ \forall\ x \in B$ (6)

Here, if all income values are N values each between v_0 and v_m, then it should be:

$\sum_{x=0}^{255} f_x = N + 1$ (7)

It is important to say that is:

$F= \{f_0, f_1,\cdots,f_{255}\}$ (8)

The set F also is related to set B, by the following:

F={f_x:x∈B} (9)

2.3. Part I Algorithm: definition and steps

In this part, the locations of the distinct value are fetched in order to build a locations table:

The algorithm is composed based on three forward procedures: *Build Lookup Table*, *Generate Output Sequence*, and *Data Compression*.

The first procedure, which called *Build Lookup Table* algorithm, builds a lookup table called *T* (Figure 2). It is similar to the adjacency list used in graphic representation.

The frequency and location table is a triple set, denoted by *(F, V, L)*. The notation $V=\{v_0, \dots v_m\}$ is the set of all possible income m ordered sequence of values could be found. All these values are equal the notation $F = \{f_0. \dots f_m\}$, is the set of all frequency of each value belong to V. The $L = \{L_0. \dots L_m\}$, is set of sets of locations where the values $v_0. \dots v_m$ are occurred, e.g. L_{100} is a set of all locations have the value 100.

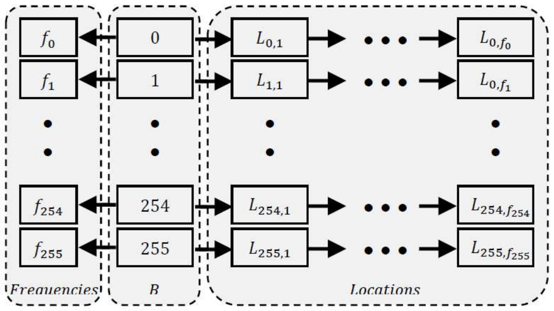


Figure 2: Lookup table T

The main column of the lookup table T contains sequentially list of all possible elements of set B as shown in figure (2). Hence, one cell for each $x \in B$. The second column, on the left side of the table T, is the values of frequency of the elements, which labeled f_x. The third part of the table T is on the right hand side of the table T, where there are 256 lists. For each $x \in B$ there is a list, called L_x. The list L_x contains of all possible values of location where the element x found in sequence S. The total number of locations where the element x found is f_x. If such a location labeled $l_{x.i}$, then $L_x = \{l_{x.1} . l_{x.2} . \dots . l_{x.f_x}\}$ .This part of table could be seen as a linked list of location values.

Since searching is repeated multiple times on the same sequence of data, we need to speed up the process by building a table for the locations of these data values. So, without losing the locations, the lookup table is work as an index to where these values are in original sequence.

The *Build Lookup Table* algorithm, as in Figure 3, comprises two loops. According to the elements of set B, the first loop will initialize for each element x in B, a frequency counter, f_x, to zero, and a location list, L_x, to null, i.e. to be empty. Since there are 256 possible elements in B, accordingly there are 256 frequency counters, and 256 list pointers. Next, the second loop will get each element e in sequence S. And, according to the element value, v_e, the frequency value, f_x, where $x = v_e$ is incremented by one. Also, the element location, l_e, added to the locations list, L_x, where $x = v_e$.

```
Algorithm Build_Lookup_Table($S$)
        For each $x \in B$
            Make for $x$ a cell in $T$
            Initialize $f_x$ to zero
            Initialize $L_x$ to null
        For each $e \in S$
            Increment $f_{x=v_e}$ by one
            Append $L_{x=v_e}$ by $l_e$
Return $T$
```

Figure 3: Build lookup table

Proposition 1: Build Lookup Table algorithm is correct. The running time is $\Theta(N)$.
Proof: For the correctness observe that the first for-loop iteratively makes a table row for each $0 \le x \le 255$, where two initialized quantities, f_x and L_x, are located. The next for-loop iteratively get each element in sequence S, increments by one a specific frequency quantity that was initialize by the first-loop, and append a specific list, which also was initialize by the first-loop, according to the testing of the value of that element. Each iteration of both loops, the algorithm updates its local data. So they are both able to perform the computation correctly.

For the complexity observe that the first for-loop is executed $\Theta(256)$ times, and the second for-loop is executed $\Theta(N)$. Totally, algorithm has a running time is $\Theta(N)$.

The next work is done by *Generate Output Sequence* algorithm (figure 4). This algorithm used the lookup table T in order to generate the output sequence, called S'. This sequence is a result of a concatenate of the four sequence *GF, RF, QL and RL,* as shown in the figure (5). So, this algorithm firstly makes these sequences and then finally joins them in one sequence.

```
Algorithm Generate_Output_Sequence($T$)
        Initialize $QF.RF.QL.RL$ to null
        For each cell $x$ in $T$
            Compute $q_x = \lfloor f_x/256 \rfloor$
            Compute $r_x = f_x - q_x \times 256$
            Append $QF$ by $q_x$
            Append $RF$ by $r_x$
            For each $l_{x.i} \in L_x$
                Compute $q_i = \lfloor l_{x.i}/256 \rfloor$
                Compute $r_i = l_{x.i} - q_i \times 256$
                Append $QL$ by $q_i$
                Append $RL$ by $r_i$
        $S' =$ concatenation of $QF.RF.QL.RL$

    Return $S'$
```

Figure 4: Generate output sequence

The *Generate Output Sequence* algorithm start by initializes each set of $QF.RF.QL.$ *and* RL to null, or make an empty set for each. Next, algorithm works top-down on table T, for each cell x in that table, it computes the quantities q_x and r_x from the frequency counter of x, f_x. These computations using the following formulas:

$$q_x = \left\lfloor \frac{f_x}{256} \right\rfloor . and\ r_x = f_x - q_x \times 256 \qquad (10)$$

Which yields two quantities in range $0 \cdots 255$. Then, the values q_x and r_x will be added to the sets QF and RF, respectively. The next step in the procedure is working with the locations list of x, L_x. Now, each location value $l_{x.i}$ in locations list of x, L_x , is converted to two quantities q_i and r_i, using the following formulas:

$$q_i = \left\lfloor \frac{l_{x.i}}{256} \right\rfloor \ and \ r_i = l_{x.i} - q_i \times 256 \qquad (11)$$

Also, we yields two quantities in range $0 \cdots 255$. Then, the values q_i and r_i will be added to the sets QL and RL, respectively.

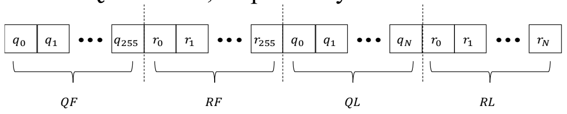


Figure 5: Generate output sequence S'

Since, the original sequence S has $N + 1$ elements, our final sequence S' (figure 5) has more elements. That is because we use two sequences QF and RF, each of length 256, and two sequences QL and RL, each of length $N + 1$. Which yields a sequence S' of totally size equal to $2 \times 256 + 2 \times (N + 1)$. When calculate the length of the two sequences, as shown in figure 6, an increasing in length of elements to double and half. Also, the ratio of the sequence S length to the sequence S' length is 1:2.0078. Therefore, we use a data compression approach to enforce a solution for the increasing length problem.

Since, our research not concern to a specific algorithm for data compression. This step is assumed as a practical work, and depends on the data that is used for testing.

$\lvert S\rvert = N + 1$	$\lvert S'\rvert = 2 \times 256 + 2(N + 1)$	$\lvert S' - S\rvert$
$= 255 \times 256 + 255+$	$= 2 \times 2^8 + 2(2^{16})$	$= 2^9 + 2^{17} - 2^{16}$
$= 255(256 + 1) + 1$	$= 2^9 + 2^{17}$	$= 2^9 + 2 \cdot 2^{16} - 2^{16}$
$= (256 - 1)(256 + 1) + 1$		$= 2^9 + 2^{16}(2 - 1)$
$= (256^2 - 1) + 1$		$= 2^9 + 2^{16}$
$= 256^2 = 2^{16}$		

Figure 1: Calculation of sequence lengths

Proposition 2: Generate Output Sequence algorithm is correct. The running time is $\Theta(N)$.

Proof: For the correctness observe that the outer for-loop iteratively uses all quantities on all 256 table T rows. In each iteration, the q_x and r_x values are computed and then update the two pointers QF and RF. The inner for-loop computes q_i and r_i for each quantity inside L_x, and then update the two pointers QL and RL. This work is always done by the iterator x which it is belong to $[0.255]$. The four list QF, RF, QL and RL, are all of a fixed length, so the process of concatenate them in one list, S', is simply and done directly when see them as pointers for specific starting location in sequence S'.The total work is:

$$\sum_{x=0}^{255} \sum_{i \in L_x} \Theta(1) = \sum_{x=0}^{255} \Theta(f_x) = \Theta(N) \qquad (12)$$

2.4. Part II Algorithm: Definition and Steps

In the opposite direction, after removing the compression, working on reproducing the original sequence S from the generated sequence S′ is simple. This procedure called Regenerate Input Sequence algorithm, as shown in figure 7. Starting with extracting the four sequences QF. RF. QL and RL from S′. This step is assumed to be strictly reversing for concatenation step in the Generate Output Sequence algorithm.

```
Algorithm Regenerate_Input_Sequence($S'$)
        Extract $QF.RF.QL.RL$ from $S'$
        Create $S$ with $N + 1$ locations
        For each $x \in B$
            Get $q_x$ from $QF$
            Get $r_x$ from $RF$
            Compute $f_x = r_x + q_x \times 256$
            Initilize counter $i$ by 1
            while $i \le f_x$
                Get $q_i$ from $QL$
                Get $r_i$ from $RL$
                Compute $l_i = r_i + q_i \times 256$
                Give value $x$ to location $l_i$ in $S$
                Increment $i$ by one
        Return $S$
```

Figure 2: Generate input sequence

Remember that $S = \langle V.L \rangle$. After done the extracting work correctly, an empty sequence S is created with $N + 1$ locations. That is, the set L is accomplished. Then we need to accomplish the set V. Remember that all element values of set V are elements of set B. The sequence S' are now an information of the distribution of set B elements within locations of sequence S. In order to fill these locations by their values, we use the same approach of take x from table T in the *Generate Output Sequence* algorithm. Accordingly, in a sequential manner, for each $x \in B$, we get q_x value from sequence QF, and get r_x from sequence RF, in order to compute the frequency counter of x, f_x, using the following formula:

$$f_x = r_x + q_x \times 256 \qquad (13)$$

This will allow us to determine how many elements in S that must be given the value x. Now, we need to determine which element locations to filled by x. So, there are f_x elements taken from the two sequence QL and RL that are of respect to the value x. Using a loop counter i to control how many time to repeat the taken elements process from the sequence QL and RL. With each value q_i taken from QL, followed by a value r_i taken from RL, a location value l_i is computed by the following formula:

$$l_i = r_i + q_i \times 256 \qquad (14)$$

Hence, the location l_i in sequence S is filled by the value of x. This work will continue until all element locations of S are filled by their values

.

Proposition 3: Regenerate Input Sequence algorithm is correct. The running time is $\Theta(N)$.
Proof: For the correctness observe that the four list QF, RF, QL and RL, are all of a fixed length, so the process of extracting them from S', is simply done by determine a starting pointer for each. In the other hand, the length of the retrieved sequence S is known be $N+1$. The outer for-loop iteratively uses $QF$and RF. In each iteration, q_x and r_x are retrieved from $QF$and RF, respectively, in concurrent way for each $0 \leq x \leq 255$, in order to compute f_x. Next, the inner while-loop, iteratively reads from QL and RL concurrently two values, q_i and r_i, respectively, to compute the location l_i, in which the current value of iterator x to be written. Here, the iterator i, $1 \leq i \leq f_x$, control when to stop reading from QL and RL.

The total time work is the same as it for Generate Output Sequence algorithm that time complexity computed as follows:

$$\sum_{x=0}^{255} \sum_{i \in L_x} \Theta(1) = \sum_{x=0}^{255} \Theta(f_x) = \Theta(N) \qquad (15)$$

Proposition 4: The time complexity for all three algorithms *Build Lookup Table, Generate Output Sequence*, and *Regenerate Input Sequence* is $\Theta(N)$, where N related to the size of sequence S.
Proof: The working time, the summation of the three algorithms running time, $\Theta(N) + \Theta(N) + \Theta(N)$, is $\Theta(N)$.

3. RESULTS AND DESCUSSION

The proposed scheme has been implemented in the C# language and Microsoft Excel 2016 with several test images. Image files are used as a source for correlated data. Some results applied on the TIF format images are below. In the histograms of the permuted elements the nearly uniform distribution of elements values is achieved. In Table 1, 10 of 26 sample tests have been experimented as image files for input sequences. The histograms of permuted sequence are shown in permutation histogram column. Also, Table 1 presents the column of the compressed permutation sequence histogram. The size in KB for original, permuted, and compressed files listed in Table 2(a). Information entropy is defined as the average amount of information produced by a source of data. Entropy is a measure of uncertainty. High entropy means the data has high variance and thus lot of information is included. The entropy measures of the original blocks and the produced blocks are shown in Table 2 (b). The correlation coefficients between the original sequence and the permutated sequence for each sample are shown in Table 2 (c). The samples are selected according to the differences in the original histogram, as illustrated in appendix I.

4. CONCLUSION

A simple to implement yet effective method has been proposed in this paper for elements permutation taken from files using element frequency and locations permutation method. The main idea is that the strong information in a file can be reduced by decreasing the correlation among the elements using element frequency and locations permutation method. This paper has presented as a method for uniform permutations for image data components. The intelligible information present in an image is due to the correlation among the image elements in a given arrangement. The experimental results have shown that the process of dividing and using frequency-location permutation of image blocks resulted in a lower correlation, a higher entropy value, and a more uniform histogram. Furthermore, the permutation process refers to the operation of dividing and replacing an arrangement of the original image, and thus the generated one can be viewed as an arrangement of blocks.

REFERENCES

[1] Pesarin F, Salmaso L. Permutation tests for complex data: theory, applications and software: John Wiley & Sons. 2010.
[2] Amishima T, Okamura A, Morita S, Kirimoto T. Permutation method for ICA separated source signal blocks in time domain. *IEEE Transactions on Aerospace and Electronic Systems*. 2010;46(2):899-904.
[3] Fu C, Lin B-b, Miao Y-s, Liu X, Chen J-j. A novel chaos-based bit-level permutation scheme for digital image encryption. *Optics communications*. 2011;284(23):5415-5423.

[4] Mitra A, Rao YS, Prasanna S. A new image encryption approach using combinational permutation techniques. *International Journal of Computer Science*. 2006;1(2):127-131.

[5] Sheri PK, Murthy AT. Advanced Image Encryption using Combination of Permutation. *International Journal of Emerging Engineering Research and Technology*. 2015;3(4):52-59.

[6] Barry WT, Nobel AB, Wright FA. Significance analysis of functional categories in gene expression studies: a structured permutation approach. *Bioinformatics*. 2005;21(9):1943-1949.

[7] Winkler AM, Ridgway GR, Webster MA, Smith SM, Nichols TE. Permutation inference for the general linear model. Neuroimage. 2014;92:381-397.

[8] Hardiya P, Gupta R. Image Encryption Based on Pixel Permutation and Text Based Pixel Substitution. *IJSRD*. 2014;2(7):142-145.

[9] Younes MAB, Jantan A. An image encryption approach using a combination of permutation technique followed by encryption. *International journal of computer science and network security*. 2008;8(4):191-197.

[10] Dattathreya MS. *System and method for handling invalid condition of a data element*. US7493334B1(Patent). 2009.

Appendix I

Table 1: The experimented samples

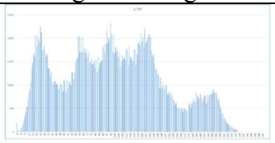
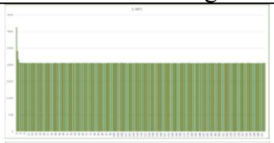
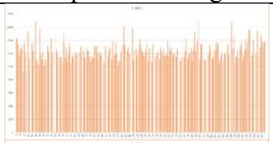
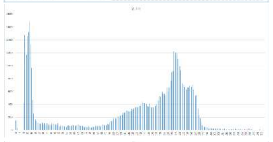
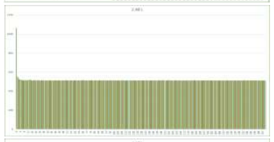
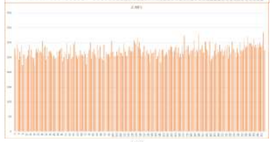
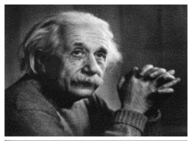
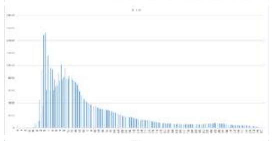
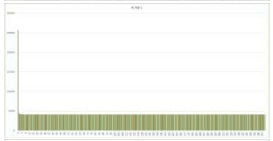
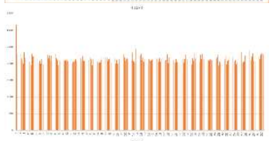
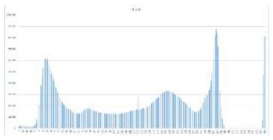
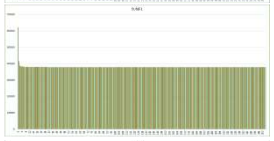
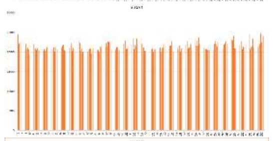
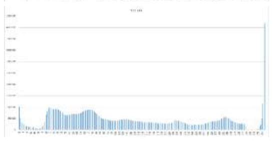
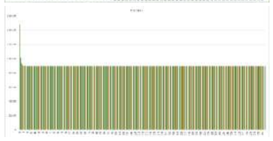
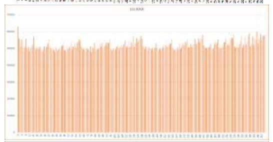
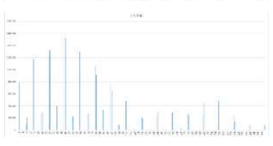
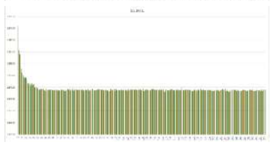
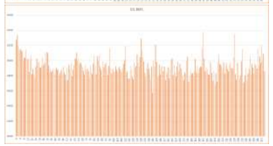
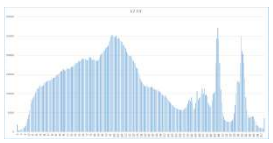
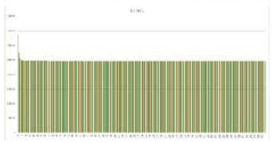
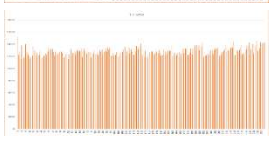
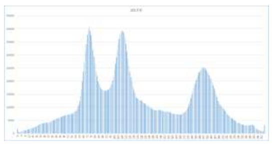
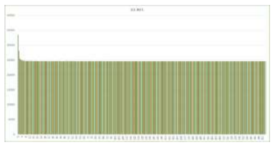
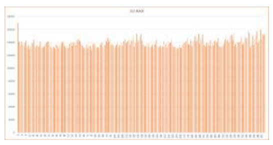
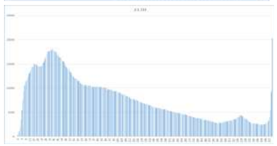
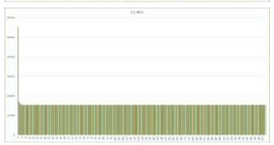
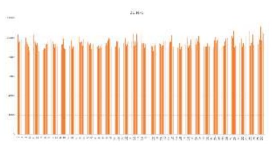
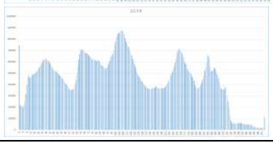
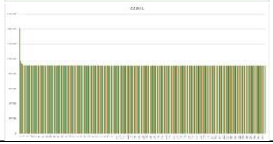
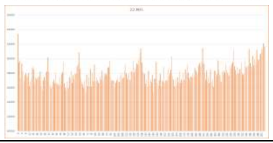
Original	Original histogram	Permutation histogram	Compressed histogram
			
			
			
			
			
			
			
			
			
			

Table 2: (a) Size in KB for orginal, permuated, and compressed files (Figure 8). (b) Entropy results for orginal elements and the corresponding permuted elements (Figure 9). (c) Correlation between original and permuted sequences (Figure 10).

	a			b		c
Image No.	Original	Permuted	Compresse d	Original	Permuted	Correlation
1	262,549	526,741	317,792	7.5992983	7.9992534	-0.1071
2	65,915	131,963	69,546	7.0264735	7.9975146	-0.0185
3	1,048,710	2,105,478	1,249,159	7.5235114	7.9996149	+0.5638
4	544,014	1,072,398	547,126	6.9972147	7.9658957	-0.0393
5	5,015,774	10,035,422	5,405,778	7.0045683	7.9964787	-0.0252
6	12,216,574	24,501,502	14,222,731	7.3143967	7.9988913	-0.0372
7	10,877,038	21,774,958	11,654,715	7.4698666	7.9973532	+0.1970
8	1,422,636	2,809,644	1,691,867	7.7626486	7.9721755	-0.1165
9	4,857,796	9,745,348	5,490,214	7.4263056	7.9990188	-0.0622
10	11,495,698	23,054,098	13,321,509	7.6220376	7.9989755	+0.0908
11	7,138,088	14,271,272	9,349,120	7.8427694	7.9960315	+0.5869
12	1,930,654	3,846,046	2,165,420	7.3014691	7.9902275	-0.0139
13	5,614,348	11,228,428	7,392,444	7.9000299	7.9964586	+0.5633
14	1,179,248	2,302,064	1,478,571	7.9133797	7.9390950	+0.4362
15	2,359,450	4,737,178	1,260,536	5.1132753	7.9984390	+0.1619
16	6,211,560	12,420,072	8,148,287	7.8877972	7.9961955	+0.5444
17	3,145,882	6,316,186	3,270,663	7.7514876	7.9996105	-0.1474
18	15,116,698	30,307,738	16,904,973	7.7172297	7.9987229	+0.4543
19	8,247,272	16,503,272	8,650,919	7.0034290	7.9964424	-0.0252
20	3,145,882	6,316,186	3,559,137	7.5589123	7.9996158	-0.1049
21	2,001,154	3,982,594	2,416,647	7.7571972	7.9888196	-0.1061

22	11,614,618	23,305,114	12,281,158	7.7910437	7.9992485	+0.0721
23	9,313,096	18,691,912	10,031,677	7.3657841	7.9992167	-0.1040
24	2,339,434	4,651,114	2,419,407	7.2934555	7.9856562	-0.0505
25	786,572	1,579,148	814,081	6.6649040	7.9992782	-0.0424
26	1,259,596	2,514,508	1,206,806	7.1949384	7.9933105	-0.0232

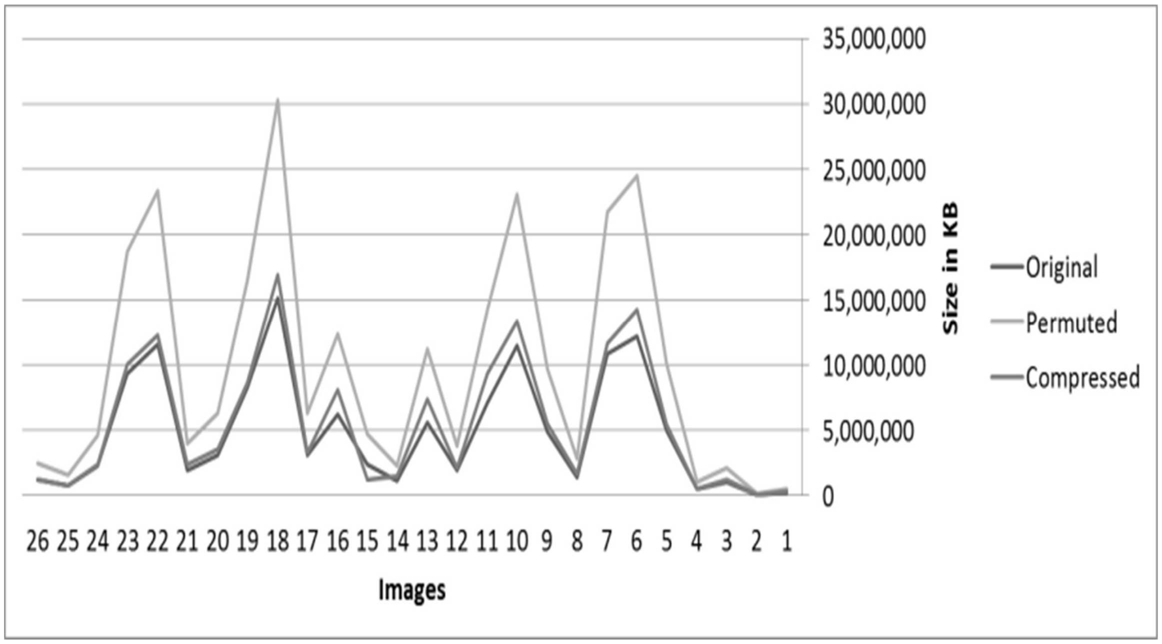


Figure 8: Size in KB for original, permuated, and compressed files