A NEW PERMUTATION METHOD FOR SEQUENCE OF ORDER $2^8$

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

1,2,3 University of Kufa, Faculty of Education for Girls, Department of Computer Science, Najaf, Iraq
E-mail: 1abdullaa.lafta@uokufa.edu.iq, 2ammar.khaleel@uokufa.edu.iq, 3safaaj.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 \( 255 \times 256 + 255 \). As illustrate in figure (1), the two sets \( V \) and \( L \) are 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 \( S \).

\[
V = \{v_0, v_1, \ldots, v_N\} \text{ and } L = \{l_0, l_1, \ldots, l_N\} \tag{1}
\]

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

\[
S = \{(v_0, l_0), (v_1, l_1), \ldots, (v_N, l_N)\} \tag{2}
\]

Define the set \( B \) is, \( B = \{x: 0 \leq x \leq 255\} \) 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. \( 0 \leq v_e \leq 255 \) for any \( e \) such that \( 0 \leq e \leq N \). That is, \( V = \{v_e: v_e \in B \forall 0 \leq e \leq N\} \).

Whereas, the locations of elements are belonging to the range \( 0 \cdots N \), i.e. \( L = \{l_e: 0 \leq l_e \leq N \forall 0 \leq e \leq N\} \). The element values are explicit information, whereas element location values are implicit information.

\[
\begin{array}{cccc}
 v_0, l_0 & v_1, l_1 & v_2, l_2 & \cdots & v_N, l_N \\
\end{array}
\]

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:

\[
QL = \{q_0, q_1, \ldots, q_N\} \text{ and } RL = \{r_0, r_1, \ldots, r_N\} \tag{3}
\]

\[
QF = \{q_0, q_1, \ldots, q_{255}\} \text{ and } RF = \{r_0, r_1, \ldots, r_{255}\} \tag{4}
\]

Such that, \( q_i \in B \forall q_i \in QL \) and \( r_i \in B \forall r_i \in RL \) for \( 0 \leq i \leq N \). Also, \( q_i \in B \forall q_i \in QF \) and \( r_i \in B \forall r_i \in RF \) for \( 0 \leq i \leq 255 \).

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

\[
l_i = q_i \times 256 + r_i \text{ for } q_i \in QL \text{ and } r_i \in RL \forall 0 \leq i \leq N \tag{5}
\]

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:

\[
f_i = q_i \times 256 + r_i \text{ where } q_i \in QF \text{ and } r_i \in RF \forall x \in B \tag{6}
\]

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

\[
\sum_{x=0}^{255} f_x = N + 1 \tag{7}
\]

It is important to say that is:

\[
F = \{f_0, f_1, \ldots, f_{255}\} \tag{8}
\]

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

\[
F = \{f_x: x \in B\} \tag{9}
\]

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 related to the set \( L \), such that:

\[
Q = \{q_0, q_1, \ldots, q_N\} \text{ and } RL = \{r_0, r_1, \ldots, r_N\} \tag{5}
\]

The set \( QL \) also is related to set \( B \), by the following:

\[
Q = \{q_x: x \in B\} \tag{6}
\]

The set \( QF \) also is related to set \( B \), by the following:

\[
Q = \{q_x: x \in B\} \tag{7}
\]

The set \( RF \) also is related to set \( B \), by the following:

\[
Q = \{q_x: x \in B\} \tag{8}
\]

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

\[
F = \{f_x: x \in B\} \tag{9}
\]
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 $L_x = \{l_{x,1}, l_{x,2}, ..., l_{x,f_x}\}$. 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 a cell in $T$
  Initialize $f_x$ to zero
  Initialize $L_x$ to null
For each $e$ in $S$
  Increment $f_{v_e}$ by one
  Append $L_{v_e}$ by $l_e$
```

**Figure 2: Lookup table $T$**

**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 $0 \leq x \leq 255$, 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' = $ concatenateation 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:

$$q_x = \lfloor \frac{f_x}{256} \rfloor \text{, and } r_x = f_x - q_x \times 256$$

(10)
Which yields two quantities in range 0...255. Then, the values $q_x$ and $r_x$ will be added to the sets $QF$, $RF$, 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:

$$q_i = \frac{l_{x,i}}{256} \quad \text{and} \quad r_i = l_{x,i} - q_i \times 256 \quad (11)$$

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

![Figure 5: Generate output sequence $S'$](image)

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.

![Figure 1: Calculation of sequence lengths](image)

**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:

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

**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](image)

Remember that $S = (V, L)$. 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:

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

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:

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

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

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

**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 permuted 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


Appendix I

<table>
<thead>
<tr>
<th>Original</th>
<th>Original histogram</th>
<th>Permutation histogram</th>
<th>Compressed histogram</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1" alt="Image" /></td>
<td><img src="image2" alt="Image" /></td>
<td><img src="image3" alt="Image" /></td>
<td><img src="image4" alt="Image" /></td>
</tr>
<tr>
<td><img src="image5" alt="Image" /></td>
<td><img src="image6" alt="Image" /></td>
<td><img src="image7" alt="Image" /></td>
<td><img src="image8" alt="Image" /></td>
</tr>
<tr>
<td><img src="image9" alt="Image" /></td>
<td><img src="image10" alt="Image" /></td>
<td><img src="image11" alt="Image" /></td>
<td><img src="image12" alt="Image" /></td>
</tr>
<tr>
<td><img src="image13" alt="Image" /></td>
<td><img src="image14" alt="Image" /></td>
<td><img src="image15" alt="Image" /></td>
<td><img src="image16" alt="Image" /></td>
</tr>
<tr>
<td><img src="image17" alt="Image" /></td>
<td><img src="image18" alt="Image" /></td>
<td><img src="image19" alt="Image" /></td>
<td><img src="image20" alt="Image" /></td>
</tr>
<tr>
<td><img src="image21" alt="Image" /></td>
<td><img src="image22" alt="Image" /></td>
<td><img src="image23" alt="Image" /></td>
<td><img src="image24" alt="Image" /></td>
</tr>
<tr>
<td><img src="image25" alt="Image" /></td>
<td><img src="image26" alt="Image" /></td>
<td><img src="image27" alt="Image" /></td>
<td><img src="image28" alt="Image" /></td>
</tr>
<tr>
<td><img src="image29" alt="Image" /></td>
<td><img src="image30" alt="Image" /></td>
<td><img src="image31" alt="Image" /></td>
<td><img src="image32" alt="Image" /></td>
</tr>
<tr>
<td><img src="image33" alt="Image" /></td>
<td><img src="image34" alt="Image" /></td>
<td><img src="image35" alt="Image" /></td>
<td><img src="image36" alt="Image" /></td>
</tr>
</tbody>
</table>

Table 1: The experimented samples
Table 2: (a) Size in KB for original, permuted, and compressed files (Figure 8). (b) Entropy results for original elements and the corresponding permuted elements (Figure 9). (c) Correlation between original and permuted sequences (Figure 10).

<table>
<thead>
<tr>
<th>Image No.</th>
<th>Original</th>
<th>Permuted</th>
<th>Compressed</th>
<th>Original</th>
<th>Permuted</th>
<th>Correlation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>262,549</td>
<td>526,741</td>
<td>317,792</td>
<td>7.5992983</td>
<td>7.9992534</td>
<td>-0.1071</td>
</tr>
<tr>
<td>2</td>
<td>65,915</td>
<td>131,963</td>
<td>69,546</td>
<td>7.0264735</td>
<td>7.9975146</td>
<td>-0.0185</td>
</tr>
<tr>
<td>3</td>
<td>1,048,710</td>
<td>2,105,478</td>
<td>1,249,159</td>
<td>7.5235114</td>
<td>7.9996149</td>
<td>+0.5638</td>
</tr>
<tr>
<td>4</td>
<td>544,014</td>
<td>1,072,398</td>
<td>547,126</td>
<td>6.9972147</td>
<td>7.9658957</td>
<td>-0.0393</td>
</tr>
<tr>
<td>5</td>
<td>5,015,774</td>
<td>10,035,422</td>
<td>5,405,778</td>
<td>7.0045683</td>
<td>7.9964787</td>
<td>-0.0252</td>
</tr>
<tr>
<td>6</td>
<td>12,216,574</td>
<td>24,501,502</td>
<td>14,222,731</td>
<td>7.3143967</td>
<td>7.9988913</td>
<td>-0.0372</td>
</tr>
<tr>
<td>7</td>
<td>10,877,038</td>
<td>21,774,958</td>
<td>11,654,715</td>
<td>7.4698666</td>
<td>7.9973532</td>
<td>+0.1970</td>
</tr>
<tr>
<td>8</td>
<td>1,422,636</td>
<td>2,809,644</td>
<td>1,691,867</td>
<td>7.7626486</td>
<td>7.9721755</td>
<td>-0.1165</td>
</tr>
<tr>
<td>9</td>
<td>4,857,796</td>
<td>9,745,348</td>
<td>5,490,214</td>
<td>7.4263056</td>
<td>7.9990188</td>
<td>-0.0622</td>
</tr>
<tr>
<td>10</td>
<td>11,495,698</td>
<td>23,054,098</td>
<td>13,321,509</td>
<td>7.6220376</td>
<td>7.9989755</td>
<td>+0.0908</td>
</tr>
<tr>
<td>11</td>
<td>7,138,088</td>
<td>14,271,272</td>
<td>9,349,120</td>
<td>7.8427694</td>
<td>7.9960315</td>
<td>+0.5869</td>
</tr>
<tr>
<td>12</td>
<td>1,930,654</td>
<td>3,846,046</td>
<td>2,165,420</td>
<td>7.3014691</td>
<td>7.9902275</td>
<td>-0.0139</td>
</tr>
<tr>
<td>13</td>
<td>5,614,348</td>
<td>11,228,428</td>
<td>7,392,444</td>
<td>7.9000299</td>
<td>7.9964586</td>
<td>+0.5633</td>
</tr>
<tr>
<td>14</td>
<td>1,179,248</td>
<td>2,302,064</td>
<td>1,478,571</td>
<td>7.9133797</td>
<td>7.9390950</td>
<td>+0.4362</td>
</tr>
<tr>
<td>15</td>
<td>2,359,450</td>
<td>4,737,178</td>
<td>1,260,536</td>
<td>5.1132753</td>
<td>7.9984390</td>
<td>+0.1619</td>
</tr>
<tr>
<td>16</td>
<td>6,211,560</td>
<td>12,420,072</td>
<td>8,148,287</td>
<td>7.8877972</td>
<td>7.9961955</td>
<td>+0.5444</td>
</tr>
<tr>
<td>17</td>
<td>3,145,882</td>
<td>6,316,186</td>
<td>3,270,663</td>
<td>7.7514876</td>
<td>7.9996105</td>
<td>-0.1474</td>
</tr>
<tr>
<td>18</td>
<td>15,116,698</td>
<td>30,307,738</td>
<td>16,904,973</td>
<td>7.7172297</td>
<td>7.9987229</td>
<td>+0.4543</td>
</tr>
<tr>
<td>19</td>
<td>8,247,272</td>
<td>16,503,272</td>
<td>8,650,919</td>
<td>7.0034290</td>
<td>7.9964424</td>
<td>-0.0252</td>
</tr>
<tr>
<td>20</td>
<td>3,145,882</td>
<td>6,316,186</td>
<td>3,559,137</td>
<td>7.5589123</td>
<td>7.9996158</td>
<td>-0.1049</td>
</tr>
<tr>
<td>21</td>
<td>2,001,154</td>
<td>3,982,594</td>
<td>2,416,647</td>
<td>7.7571972</td>
<td>7.9888196</td>
<td>-0.1061</td>
</tr>
<tr>
<td></td>
<td>Original</td>
<td>Permuted</td>
<td>Compressed</td>
<td>Entropy Original</td>
<td>Entropy Permuted</td>
<td>Entropy Compressed</td>
</tr>
<tr>
<td>---</td>
<td>---------</td>
<td>---------</td>
<td>------------</td>
<td>-----------------</td>
<td>-----------------</td>
<td>-------------------</td>
</tr>
<tr>
<td>22</td>
<td>11,614,618</td>
<td>23,305,114</td>
<td>12,281,158</td>
<td>7.7910437</td>
<td>7.9992485</td>
<td>7.9992485</td>
</tr>
<tr>
<td>23</td>
<td>9,313,096</td>
<td>18,691,912</td>
<td>10,031,677</td>
<td>7.3657841</td>
<td>7.9992167</td>
<td>7.9992167</td>
</tr>
<tr>
<td>24</td>
<td>2,339,434</td>
<td>4,651,114</td>
<td>2,419,407</td>
<td>7.2934555</td>
<td>7.9856562</td>
<td>7.9856562</td>
</tr>
<tr>
<td>25</td>
<td>786,572</td>
<td>1,579,148</td>
<td>814,081</td>
<td>6.6649040</td>
<td>7.9992782</td>
<td>7.9992782</td>
</tr>
<tr>
<td>26</td>
<td>1,259,596</td>
<td>2,514,508</td>
<td>1,206,806</td>
<td>7.1949384</td>
<td>7.9933105</td>
<td>7.9933105</td>
</tr>
</tbody>
</table>

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