Modern Applied Science; Vol. 12, No. 11; 2018
ISSN 1913-1844E-ISSN 1913-1852
Published by Canadian Center of Science and Education

CRUSH: A New Lossless Compression Algorithm

Evon Abu-Taieh[1] & Issam AlHadid[1]

[1] Faculty of Computer Information Systems, The University of Jordan, Aqaba, Jordan

Correspondence: Evon Abu-Taieh, Faculty of Computer Information Systems, The University of Jordan, Aqaba, Jordan. E-mail: abutaieh@gmail.com

Received: May 25, 2018 Accepted: September 20, 2018 Online Published: October 29, 2018

doi:10.5539/mas.v12n11p406 URL: https://doi.org/10.5539/mas.v12n11p406

Abstract

Multimedia is highly competitive world, one of the properties that is reflected is speed of download and upload of multimedia elements: text, sound, pictures, animation. This paper presents CRUSH algorithm which is a lossless compression algorithm. CRUSH algorithm can be used to compress files. CRUSH method is fast and simple with time complexity O(n) where n is the number of elements being compressed.Furthermore, compressed file is independent from algorithm and unnecessary data structures. As the paper will show comparison with other compression algorithms like Shannon–Fano code, Huffman coding, Run Length Encoding, Arithmetic Coding, Lempel-Ziv-Welch (LZW), Run Length Encoding (RLE), Burrows-Wheeler Transform.Move-to-Front (MTF) Transform, Haar, wavelet tree, Delta Encoding, Rice &Golomb Coding, Tunstall coding, DEFLATE algorithm, Run-Length Golomb-Rice (RLGR).

Keywords: compression,variable length compression, statistical compression, Huffman cod

1. Introduction

Multimedia is highly competitive world, one of the properties that is reflected is speed of download and upload of multimedia elements: text, sound, pictures, animation. Especially in this day and age where multimedia is an essential part in the internet world. Speed of download, uploaded depends mainly on the size of the file, no matter what the file format. Compression allows the users to reduce the size of the file. Other methods, multimedia, resort to is interlacing and interleaving.All methods, compression, interlacing and interleaving are used to shorten the time and speed up the showing of the file. Compression algorithms are not new some go back to 1910 like HAAR WAVELET TRANSFORM and others are as new as 2011 LZ4. Hence, there is an essential need to think of more compression algorithm that can perform better than the existing compression algorithms.This research suggests a new compression algorithm named CRUSH short for Compression Up Shapes. CRUSH (Compression Up Shapes) is simple, fast and with time complexity O(n) where n is the number of elements being compressed. CRUSH performs better than Huffman and Shannon-Fano algorithm as will be shown in the paper. The storage complexity of CRUSH is also O(n) where n is the number of elements to be compressed.

The paper first presents 12 compression algorithms that are related to the main theme of the paper.Then the paper presents the suggested algorithm CRUSH and further explains the algorithm using pseudo code.Next the paper describes the compression file, the output of the compression algorithm CRUSH, to further explain the algorithm the paper shows a working example of the algorithm at work. Then the paper discusses the time and storage complexity of CRUSH.Next the paper shows the compression ratio of CRUSH in comparison with Huffman code, Shannon-Fano algorithm. Then the paper presents discussion of the advantages and disadvantages of CRUSH. The paper then discusses application of CRUSH and future work that may improve the work of CRUSH.

2. Related Studies

The researcher conducted a study which produced a paper titled "The Pillars of Lossless Compression Algorithms a Road Map and Genealogy Tree" (Abu-Taieh, 2018). The tree showed the interrelationships between the 40 algorithms.The author duped 12 algorithms as the pillars because they were original and unique methods.The author explained each one with the algorithm and example for each algorithm. The 12 algorithms are: Shannon–Fano code, Huffman coding, Run Length Encoding, Arithmetic Coding, Lempel-Ziv-Welch (LZW), Run Length Encoding (RLE), Burrows-Wheeler Transform. Move-to-Front (MTF) Transform, Haar,

wavelet tree, Delta Encoding, Rice &Golomb Coding, Tunstall coding, DEFLATE algorithm, Run-Length Golomb-Rice (RLGR).

```
1   Horizontal segmentation
        For each val image matrix                       // there are 3 val matrices
                Scan image
                Default_color=color with maximum frequency
                5   Counter=1
                For row=1 to image_rows                 //read matrix row by row
                    For column=1 to image_cols
                      Array[counter]=image[row,column]
                      If Array[counter]<>Default_color then      // find segment's beginning and   end
                          10   If Array[counter] <> Array[counter+1] then set start_sgment=counter
                          If Array[counter+1] <> Array[counter] then set End_sgment=counter
                          Segment_color= image [row,column]
                      Counter=counter +1
                Sort segments according to segment color
                15   Open text file                     //create the compression file
                For row =1 to end of segments table
                        Insert segment-color followed by semicolon
                        Insert all segment start & end
        Calculate Horizontal segmentation compression Ratio
20  Vertical segmentation
        For each val image matrix
                Scan image
                Default_color=color with maximum frequency
                Counter=1
```

Figure 1. CRUSH the suggested algorithm

SHANNON–FANO CODE developed in 1948-1949; the essence of the code is counting the frequency of each symbol in the file. Then, divide the frequency into two. And code into a balance binary tree the symbols of the file (Fano, 1949)(Shannon, 1948)(Abramson, 1963).In 1973 (Cover, 1973) published another version named Enumerative Coding. In 1976 (Rissanen, 1976) introduced Last in First Out (LIFO) version of the algorithm. In 1976 Pasco (Pasco, 1976) introduced the FIFO version of the algorithm. In 1979 "stream" code was discovered by Rubin (1979) as an improvement of Pasco's work. Martin (1979) and Jones (1981) developed P-based FIFO arithmetic codes (Jones, 1981) and (Martin, 1979).

HUFFMAN CODE developed in 1952 and uses again the statistical coding where most frequently used symbol is coded on the shorter branch of the tree hence symbol is coded with the shortest binary code.The tree in Huffman is not balanced binary tree Codes (Huffman, 1952).

The LZ compression method was developed by Jacob Ziv and Abraham Lempel in 1977. The method was so interesting it bread more than 26 versions over the years. The basic philosophy of the method is to add to the dictionary new sequence of letters.The method excels and thrives on large files(Cslearning, 2013)(Storer& Szymanski, 1982).

Run Length Encoding (RLE) was used since 1967 for TV signals compression and excels on sorted symbols (Abu-Taieh, 2018).Burrows- Wheeler Transform (BWT) is also very simple and best used with RLE (Burrows & Wheeler, 1994). Move To Front (MTF) Transform by (Ryabko, 1980). MTF is very simple and easy to use.Haar Wavelet Transform was developed in 1910 by AlfrédHaar the basic philosophy is two formulas two unknown the

famous calculus problem.The wavelet Tree developed in 2003 in a paper (Gross, Gupta, & Vitter, 2003) depends on the counting of the compressed sequence and the frequency of the symbols.

Delta Encoding is known as Delta compression and Data Differencing. The method stores the difference rather the data itself. Rice &Golomb Coding developed in (Golomb, 1966)(Rice, 1979) based on the idea of breaking the number into quotient and remainder. Tunstall coding was developed in 1967 by Brian Parker Tunstall.Tunstall coding builds a tree for all possible combinations of the symbols used in a text.The tree is built according the frequency of the symbols hence the symbol with highest probability is to branch out in the tree (Abu-Taieh, 2018).

The hybrid type of algorithms like DEFLATE algorithm and Run-Length Golomb-Rice (RLGR). Both are mix of a algorithms. Deflate is basically LZ77 and LZSS and Huffman Code. On the other hand Run-Length Golomb-Rice as the name suggests is RL and Golomb-Rice (Malvar, 2006).

3. The Suggested Algorithm: CRUSH

The input of the algorithm will be three matrices as in mathlab: val(:,:,1),val(:,:,2), val(:,:,3). Each matrix is designated to a color red, blue, and green.The algorithm first segments the image to horizontal segmentation in other words row by row. Then the algorithm will segment the image vertically or column by column. Then in line 19 calculates the horizontal compression ratio and in line 38 the vertical compression ratio will be calculate.In line 40 the both ratios are compared and the better one is used and compressed file is used accordingly.

The algorithm will scan the matrix and designate a default color. The default color is the color with highest frequency. The algorithm will scan each val matrix and convert it to one dimensional array, in preparation to break the color into continuous segments. The algorithm will produce a table with each segment color and *start* of the segment and *end* of segment indices. Then the algorithm will proceed to sort the segments table according to segment color. The sorted table will be injected in a file with color separated by semicolon and followed by *start* and *end* of segments. The organization of the compression file will be explained in details. Next, the algorithm will be explained line by line.

The algorithm has two main sections: horizontal and vertical segmentation. The first section segments the matrices horizontally row-by-row. The second section of the algorithm which starts at (line 20), segments the matrices column-by-column.The line 2 is a for loop that will compress horizontally the 3 *val* matrices. In line 3 the image is scanned to find the dominating color, which the color with the highest frequency.Lines 6 and 7 are nested loop to read the matrix and convert it to single dimension array. The line 10 and 11 are to find the beginning and the end of the segment.The line 14 is to sort according to color index all the segments. Then in line 15 the compression file is opened which is text file, and the lines 16 and 17 will append to the file the segments color and the *begin* and *end* of the segment.The algorithm in line 19 will calculate the compression ratio for the horizontal segmentation compression.The next section in the algorithm lines 20 to 38 are the same as the first section with exception that in lines 25 and 26 the matrices are read column-by-column.Line 39 compares which segmentation is better, comparing the horizontal segmentation compression ratio to the vertical segmentation compression ratio.

3.1 Description of Compression File

In this section the content of the compressed file will be explained in details. The compressed file is a simple text file that includes width or height of the image, type of compression weather horizontal or vertical, the default color and the segments color, start, and end.Figure (3) show the content of the compressed file with a working example. The compressed file will include the following: *Width or height of image:*it is essential to keep either the width or height of the image.One can be driven from the other since the length of the one-dimensional array is kept.

3.2 Type of Compression: Vertical or Horizontal.

The default color: the default color is the color with highest frequency, since that number is used then there is no need to keep all the segments data.Hence, the algorithm will keep information (start and end of segment) and the rest will be filled by the default color when the file decompressed.

Segment-color, start, and *end*: The segment color and the start and end of each segment must be included in the compressed file.Color is delimited by semi-colon and start and end delimited by commas.

3.3 Working Example

In this section a working example of the CRUSH algorithm working on compressing a matrix taken from an

image.The image is shown in figure (2), and then the section will explain the frequency of colors tables seen in table 1, and the segments tables seen in table 2, 3, and 4, and the compressed file shown in figure 3.

A 41X41 pixel BMP picture is made of three matrixes each with 41 rows and 41 columns. The three matrixes are duped mathlab as val(:,:,1),val(:,:,2), val(:,:,3). Each cell has a value range 0 to 255 (one byte) hence each matrix has 41*41*1 byte which equals 1681 bytes. Therefore, the three matrices have 41X41X3X1 bytes which equals 5043 bytes hence when looking at the file through the Windows operating system one can see the size of the file is 5.1 KB.

	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13	C14	C15	C16	C17	C18	C19	C20	C21	C22	C23	C24	C25	C26	C27	C28	C29	C30	C31	C32	C33	C34	C35	C36	C37	C38	C39	C40	C41
R1	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255
R2	255	255	255	255	255	255	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255
R3	255	255	255	255	255	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	255	255	255	255	255	255
R4	255	255	255	255	255	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	255	255	255	255	255
R5	255	255	255	255	255	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	255	255	255	255	255
R6	255	255	255	255	255	255	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	255	255	255	255	255
R7	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	255	255	255	255	255	255
R8	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255
R9	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255
R10	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255
R11	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255
R12	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255
R13	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255
R14	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255
R15	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255
R16	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255
R17	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255
R18	255	255	255	255	255	255	255	255	255	255	255	255	255	237	237	237	237	237	237	237	237	237	237	237	237	237	237	237	237	237	237	237	237	237	237	237	237	237	237	237	237
R19	255	255	255	255	255	255	255	255	255	255	255	255	237	237	237	237	237	237	237	237	237	237	237	237	237	237	237	237	237	237	237	237	237	237	237	237	237	237	237	237	237
R20	255	255	255	255	255	255	255	255	255	255	255	255	237	237	237	237	237	237	237	237	237	237	237	237	237	237	237	237	237	237	237	237	237	237	237	237	237	237	237	237	237
R21	255	255	255	255	255	255	255	255	255	255	255	255	237	237	237	237	237	237	237	237	237	237	237	237	237	237	237	237	237	237	237	237	237	237	237	237	237	237	237	237	237
R22	255	255	255	255	255	255	255	255	255	255	255	255	255	237	237	237	237	237	237	237	237	237	237	237	237	237	237	237	237	237	237	237	237	237	237	237	237	237	237	237	237
R23	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255
R24	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255
R25	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255
R26	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255
R27	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255
R28	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255
R29	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255
R30	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255
R31	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255
R32	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255
R33	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255
R34	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255
R35	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255
R36	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	34	34	34	34	34	34	34	34	34	34	34	34	34	34
R37	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	34	34	34	34	34	34	34	34	34	34	34	34	34	34	34
R38	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	34	34	34	34	34	34	34	34	34	34	34	34	34	34	34
R39	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	34	34	34	34	34	34	34	34	34	34	34	34	34	34	34
R40	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	34	34	34	34	34	34	34	34	34	34	34	34	34	34
R41	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255

Figure 2. The matrix Val(:,:,1) produced for the picture using mathlab

The suggested algorithm looks at the first matrix and scans the frequency of each color: for value are found 0, 34,237, and 255. With the following frequencies in table (1):

Table 1. Frequency of colors

Cell Value	Frequency
0	143
34	73
237	143
255	1313

Since 255 is the most frequently used we forgo this value for now and we treat the cells consecutively. The reading of the matrix is done row by row. In a later section a column by column will be conducted, and reading the matrix diagonally.

We start with cells of value 0 we find that cells numbers 47 through 62 are of value 0, and cells 87 through 117, also 128 through 159, 169 through 200 and 211 through 241, and 266 through 281. As follows in table (2):

Table 2. Color (0) segments

Cell Value	From	To
0	47	62
	87	117
	128	159
	169	200
	211	241
	266	281

Again, we start with cells with the color value 237, the matrix is scanned consecutively and the following is found in table (3): cells 710 through 738 are of value 237, 750 to 779, 791 to820, 832 to 861, and 874 to 902.

Table 3. Color (237) segments

Cell Value	From	To
237	710	738
	750	779
	791	820
	832	861
	874	902

Next, we scan the matrix for the value 34; we find that cells numbers 710 through 738 are of value 34, and cells 1502 through 1517, also 1543 through 1558, 1584 through 1599 and 1626 through 1640. As follows in table (4):

Table 4. Color (34) segments

Cell Value	From	To
34	1462	1476
	1502	1517
	1543	1558
	1584	1599
	1626	1640

The cell value 255 is used as the fourth color in this matrix and there is no need to keep track of it.Each from/to indices can be represented by two bytes.Hence, the three value (0, 34, 237) and their indices can be represented by 4 bytes for each from/two and there are 6 for the color value of 0, 5 for the color value 237 and 5 for the color value 34.Furthermore, the total number of bytes is 4*6+4*5+4*5=4*16=64 bytes.

The 255 is the filler for the matrix and can be treated as such.So instead of storing 41X41X1 bytes we store only 64 bytes.The saving ration of storage is 64: 1681, in other words we save up to 1-64/1681=.9619 or almost 96% savings.

To apply the same for the other two matrixes one can multiply 64 by the number of matrix which is equals 64*3=193 bytes. Again, we calculate the savings ratio 193:(1681X3) which is almost 1- 193/5043=.9617 or 96.17%.

The output compressed file shown in figure 3 is as explained above composed of 4 main elements: the width of the image, the compression type here is *h* which stands for horizontal compression, the default color is 255. The segments are last, each color separated by a semi-colon and the segments *begin* and *end* are separated by commas.

Width=41
Compression=h
Default color=255
0;47,62,87,117,128,159,169,200,211,241,266,281,237;710,738,750,779,791,820,832,861,874,902,
34;1462,1476, 1502, 1517, 1543,1558, 1584,1599, 1626,1640.

Figure 3. The compressed file output of CRUSH algorithm

3.4 Time and Storage Complexity of CRUSH

CRUSH algorithm time complexity as seen in figure 1 is O (n) where *n* is the number of elements in the matrix. The storage complexity is O(n), where n is the number of elements being compressed. The one dimensional array used to calculate the segmentation attributes. To compare with time complexity of: Shannon–Fano Code time complexity is O (n + |symbols| * log| symbols |), Arithmetic Codingtime complexity isO(|symbols|+n), Huffman Codetime complexity isO (|symbols| log |symbols|), LZtime complexity isO(n), Run Length Encoding (RLE) time complexity isO(n), Burrows-Wheeler Transform time complexity isO(n log n) (Lippert, Mobarry, &Walenz, 2005), Move To Front (MTF)Transform time complexity is O(n), Haar Wavelet transform time complexity isO(n), Wavelet Tree time complexity isO (n log symbols), Delta Encoding time complexity isO(n), and Tunstall Coding time complexity isO(symbols log symbols).

3.5 Comparison between Huffman Code, Shannon-Fano Algorithm and CRUSH

This section will present comparison of the suggested algorithm CRUSH with two most famous compression algorithms: Huffman Coding and Shannon-Fano algorithm. The research will use both algorithms to compress the same image used in the example above to show the capabilities of the CRUSH algorithm. First, Using Huffman code to compress the above file, as explained in (Abu-Taieh, 2018), the following is conducted.

Table 5. Huffman code compression

Cell Value	Frequency	Binary Coding	Bits
0	143	111	429
34	73	110	219
237	143	10	286
255	1313	0	1313
Total			2247 = 280 bytes

First as seen in table 5, the frequencies of the value are set up, then using Huffman tree the binary coding of each cell value is driven. The cell value is coded to three bits and the value 34 is coded to three bits according to the

tree frequency, then the value 237 is coded into two bits and the value 255 is coded into one bit. Hence, the number of bits used to compress the file is (143*3+73*3+143*2+1313*1= 2247 bits). The Huffman coding will reduce the file from 1681 bytes to 2247 bits almost 280 bytes. Furthermore, compressing the file by almost 83%.

Next the paper shows the compression of Shannon-Fano algorithm to compress the same example above. The cell values are sorted ascending according to their frequencies. Then a tree is built using the least frequency as explained in (Abu-Taieh, 2018). The cell value 34 will be coded to 3 bits and the cell value 0 will be coded to 2 bits and the cell value 237 will be coded to three bits, and the cell value 255 will be coded to 1 bit, see table 6. Hence, the number of bits used to compress the file is (143*2+73*3+143*3+1313*1= 2247 bits). Shannon-Fano algorithm will reduce the file from 1681 bytes to 2247 bits almost 280 bytes. Which is very similar to Huffman Code and the compression rate is almost 83%.

Table 6. Shannon-Fano Algorithm Compression

Cell Value	Frequency	Binary Coding	Bits
34	73	110	219
0	143	10	286
237	143	111	429
255	1313	0	1313
			2247 = 280 bytes

One can reach the conclusion that CRUSH compresses more than Huffman coding and Shannon-Fano algorithm based on the simple comparison shown below, table (7):

Table 7. Comparison between Huffman, Fano, and CRUSH

Compression Method	Compressed Bytes of (41X41) Bytes	Compression Rate
CRUSH	64	64/1681=96.19%
Huffman coding	280	280/1681=83.34%
Shannon-Fano algorithm	280	280/1681=83.34%

Hence CRUSH has much better compression ratio. Furthermore both algorithms depend of the tree structure which will entail sending the tree structure of table with the compressed file, CRUSH does not need to send any lookup table or tree structure with compressed file. In order to, uncompress the file all needed is the algorithm itself.

3.6 Advantages & Disadvantages of CRUSH

CRUSH performs best when there are big spaces of the same color. And performs worst when the pixels are color-fragmented (each pixel has its own color). When compared to other compression algorithms CRUSH does not need to accompany the compressed file with lookup table nor tree. Hence the compression algorithm CRUSH can outperform both algorithms in two aspects: compression ratio and the space complexity.

4. Applications of CRUSH

Speed and accuracy are two essential ingredients in the computer world, especially in this day and age. Both, speed and time are very much related to files size. Files are made of bits and bytes and file size are measured by them. Compression reduces files size; hence, files will take less storage and faster network speed, furthermore less transfer time. When a files is transferred over networks the size of the file is an important element which reflected on the speed and time of transfer. The relation is best explained: the lighter the load the faster the transfer. Consequently, the size of file, is reflected on time of transfer and the speed of the transfer. Compression supposed to reduce the file size, and indirectly increase the speed and reduce the time needed. Transfer of file overt the network can be send/receive or upload/download times which is reflected as network total speed.

Accuracy is the second ingredients in this picture. The reduction of the files size should not affect the accuracy

of the files. One should not sacrifice the accuracy for speed. CRUSH, the suggested algorithm in this research, offers the two ingredients. CRUSH offers a high compression rate with lossless property.

5. Future Work

Many suggestions for future work can be discussed. One of the suggestions is to have the matrices read many times: horizontal, vertical and diagonal. Furthermore, the algorithm will calculate the highest compression rate and compress the image accordingly. Another suggestion is using CRUSH with delta encoding method.

6. Conclusion

The quest for a perfect compression algorithm is an open ended question in the computer science arena. This paper suggested a compression algorithm named CRUSH. The researchers tried to avoid the problems of other algorithms: compression ratio, time complexity, and storage complexity. Hence the suggested algorithm, CRUSH, is simple with running time of O(n) and storage complexity is O(n) where n is the number of elements needs to compressed. Furthermore CRUSH excels in the compression ratio and in images with homogenous colors the compression ratio is 97% unlike Huffman coding and Shannon-Fano algorithm which compression ratio were 83%, using the same image.

The paper first presents a number of 12 compression algorithms that are related to the main theme of the paper. Then the paper presents the suggested algorithm CRUSH and further explains the algorithm using pseudo code. Next the paper describes the compression file, the output of the compression algorithm CRUSH, to further explain the algorithm the paper shows a working example of the algorithm at work. Then the paper discusses the time and storage complexity of CRUSH. Next the paper shows the compression ratio of CRUSH in comparison with Huffman code, Shannon-Fano algorithm. Then the paper presents discussion of the advantages and disadvantages of CRUSH. The paper then discusses future work that may improve the work of CRUSH.

References

Abramson, N. (1963). *Information theory and coding.* New York: McGraw-Hill Book Co., Inc.

Abu-Taieh, E. (2018). The pillars of lossless compression algorithms a road map and genealogy tree. *International Journal of Applied Engineering Research, 13*(6), 3296-3414.

Burrows, M., & Wheeler, D. J. (1994). *A block sorting lossless data compression algorithm.* Digital Systems Research Center. Palo Alto, California: Digital Equipment Corporation.

Cover, T. M. (1973). Enumerative Source Coding. *IEEE Transactions on Information Theory, 19*(1), 73 - 77.

Cslearning. (2013, Dec 19). *Lempel-Ziv-Welch compression algorithm – tutorial.* Retrieved from https://www.youtube.com/watch?v=j2HSd3HCpDs

Fano, R. (1949). *The transmission of information.* MASSACHUSETTS INSTITUTE OF TECHNOLOGY. Cambridge (Mass.), USA: Research Laboratory of Electronics at MIT.

Golomb, S. (1966). Run-length encodings. *IEEE Transactions on Information Theory, 12*(3), 399-401.

Gross, R., Gupta, A., & Vitter, J. (2003). High-order entropy-compressed text indexes. *SODA '03 Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms* (pp. 841–850). Baltimore, Maryland: Society for Industrial and Applied Mathematics.

Huffman, D. (1952). A Method for the Construction of Minimum-Redundancy Codes. *Proceedings of the IRE., 40*(30), 1098–1101. IEEE.

Jones, C. B. (1981, May C. B., IT-27,280-291 (May 1981)). An efficient coding system for long source sequences. *IEEE Trans. Info. Theory*, 280-291.

Lippert, R., Mobarry, C., & Walenz, B. (2005, October). A space-efficient construction of the burrows wheeler transform for genomic data. *Journal of Computational Biology, 12*(7), 943-951.

Malvar, H. (2006). Adaptive run-length/Golomb-Rice encoding of quantized generalized Gaussian sources with unknown statistics. *DCC '06 Proceedings of the Data Compression Conference* (pp. 23-32). Washington, DC, USA: IEEE Computer Society .

Martin, G. N. (1979). Range encoding: An algorithm for removing redundancy from a digitized message. *Video and Data Recording Conference.* Southampton, England, presented at the, July.

Pasco, R. (1976). Source coding algorithms for fast data compression. *Ph.D. Thesis*. CA, USA: Department of Electrical Engineering, Stanford University.

Rice, R. F. (1979). *Some practical universal noiseless coding techniques.* California Institute of Technology . Pasadena: Jet Propulsion Laboratory.

Rissanen, J. (1976, May). Generalized kraft inequality and arithmetic coding. *IBM Journal of Research and Development, 20*(3), 198-203.

Rubin, F. (1979, Nov). Arithmetic stream coding using fixed precision registers. *IEEE Transactions on Information Theory, 25*(6), 672-675.

Ryabko, B. Y. (1980). Data compression by means of a "book stack". *Problems of Information Transmission, 16*(4), 265-269.

Shannon, C. (1948, July). A mathematical theory of communication. *The Bell System Technical Journal, 27*, 379-423.

Storer, J. A., & Szymanski, T. G. (1982, Oct). Data compression via textual substitution. *Journal of the ACM (JACM), 29*(4), 928-951.