Comparison of Text Data Compression Using Yamamoto Recursive Codes and Punctured Elias Codes

S M Hardi¹*, M R Putra¹, J T Tarigan¹ and I Jaya¹

¹ Faculty of Computer Science and Information Technology, Universitas Sumatera Utara, Jl. Universitas No. 9-A, Medan 20155, Indonesia.

*Email: vani.hardi@usu.ac.id

Abstract. This research aims to study the efficiency of two variants of variable-length codes (i.e., Yamamoto Recursive codes and Punctured Elias codes) in compressing texts. The parameter being examined are the ratio of compression, the space savings, and the bit rate. As a benchmark, all of the original (uncompressed) texts are assumed to be encoded in American Standard Codes for Information Interchange (ASCII). The overall result shows that the Punctured Elias codes are consistently more efficient to compress texts than Yamamoto Recursive codes.

1. Introduction

Data compression is the process of converting an input data stream (the source stream or the original raw data) to a new stream of data with smaller size [1]. In modern computing systems, characters or symbols are usually encoded in ASCII. Each symbol that appears on a computer screen has a different ASCII code. Since the length of each ASCII code is 8, there are 28 unique symbols in the ASCII table. Data compression can be divided into two types: lossless and lossy compressions. In lossless compression, the compressed data can be reconstructed back to the original data. Therefore, lossless compression is more suitable for text data. On the other hand, in lossy compression, some information is loss due to some methods used within the algorithms. which makes lossy compression is suitable to compress multimedia data (such as animations, audio, images, and video) [1, 3].

2. Method

In this research, we want to compare the performance of Punctured Elias codes and Yamamoto Recursive codes in term of text compression. The parameter used to analyze the result are the ratio of compression, the space savings, and the bit rate. Here is the explanation about the method.

2.1. Punctured Elias codes.

Punctured Elias codes was developed by Peter Fenwick [2] to improve the performance of Burrows-Wheele. the term “punctured” comes from the field of error control codes (ECC). If some check bits are removed, to shorted the codeword, the resulting code is referred as punctured [4].
To generate Punctured Elias codes \([4]\), suppose that we have binary value of \(n\), reverse its bits so its rightmost bit is now 1, and prepend flags for the bits of \(n\). For each bit of \(n\), create a flag of 0, except for the last (rightmost) bit (which is a 1), whose flag is 1. Prepend the flags to the reversed \(n\) and remove the rightmost bit. Thus, \(13 = 1101\) is reversed to yield 1011. The four flags 0001 are prepended and the rightmost bit of 1011 is removed to yield the final gamma code 0001|101.

### 2.2. Run-Length Encoding Algorithm

The Yamamoto Recursive codes are explained as follows \([1]\). Suppose that we have delimiter \(f\)-bit (which is a positive integer), for example 2 and 00. Generate \(B_{a,f}(n)\) table by sorting binary combination that does not start from delimiter, which include the two 1-bit strings 0 and 1; the three 2-bit strings 01, 10, and 11 and so on. Next step is to check if the list contains a combination of \(B_{a,f}(n)\) followed by another \(B_{a,f}(n)\) that accidentally contain the pattern of delimiter, for example \(B_{100,3,f}(5)\) followed by \(B_{100,3,f}(7)\) become the string 10|000. With \(B_{a,f}(n)\) table, given a positive integer \(n\), we start with \(B̅_{a,f}(n)\). We prepend to it \(B_{a,f}(n-1)\) until \(B_{a,f}(1)\), after that prepend delimited at the end of the code.

### 3. Result and Discussion

The results of this research are the process of compression and decompression has been done with strings of two different types, namely strings with the same character (Homogeneous String) and a string of several types different characters (Heterogeneous String).

#### 3.1 Homogeneous String Testing

The results of the homogeneous String test are processed by the Yamamoto Recursive Code algorithm and Punctured Elias Code algorithm can be seen in Table 1 and Table 2.

**Table 1.** Experiment Results of Homogeneous Strings with Yamamoto Recursive Code.

| Uncompressed (bits) | Char | Compressed (bits) | Compression Ratio | Bitrate (bits/symbol) | Space Saving | Compression time | Decompression |
|--------------------|------|-------------------|-------------------|-----------------------|--------------|------------------|--------------|
| 500.000            | 1    | 187502            | 0.38              | 187502                | 62%          | 36 ms            | 88 ms        |
| 1.000.000          | 1    | 375002            | 0.38              | 375002                | 62%          | 75 ms            | 158 ms       |
| 1.500.000          | 1    | 562602            | 0.38              | 562502                | 62%          | 110 ms           | 217 ms       |
| 2.000.000          | 1    | 750002            | 0.38              | 750002                | 62%          | 159 ms           | 306 ms       |
| 2.500.000          | 1    | 937502            | 0.38              | 937502                | 62%          | 189 ms           | 375 ms       |

**Table 2.** The Experiment Results of Homogeneous String Testing with Punctured Elias Code.

| Uncompressed (bits) | Char | Compressed (bits) | Compression Ratio | Bitrate (bits/symbol) | Space Saving | Compression time | Decompression |
|--------------------|------|-------------------|-------------------|-----------------------|--------------|------------------|--------------|
| 500.000            | 1    | 187502            | 0.38              | 187502                | 62%          | 36 ms            | 88 ms        |
| 1.000.000          | 1    | 375002            | 0.38              | 375002                | 62%          | 75 ms            | 158 ms       |
| 1.500.000          | 1    | 562602            | 0.38              | 562502                | 62%          | 110 ms           | 217 ms       |
| 2.000.000          | 1    | 750002            | 0.38              | 750002                | 62%          | 159 ms           | 306 ms       |
| 2.500.000          | 1    | 937502            | 0.38              | 937502                | 62%          | 189 ms           | 375 ms       |

Based on the test results of Table 4.6 and Table 4.7 we can make a comparison chart of the results Homogeneous String testing Yamamoto Recursive Code algorithm and algorithm Punctured Elias Code based on the variable Bitrate, Compression Ratio (CR), Space Savings, Compression Time as in Figure 1, Figure 2, Figure 3, Figure 4 and Figure 5.
Based Figure 1 the Bitrate Graph can be concluded that the algorithm Punctured Elias Code is more efficient in terms of bitrate compared to Yamamoto Recursive Code algorithm. Due to the value of the Bitrate owned by Punctured Elias Code algorithm is lower.

From Figure 2 Compression Ratio Graph it can be concluded that the algorithm is Punctured Elias Code is more efficient in terms of Compression Ratio compared to Yamamoto Recursive Code algorithm. Because the algorithm that has the lowest Cr value, the algorithm is better used for the process compress file, in this case a homogeneous string file.

From Figure 3 the Space Saving graph can be concluded that the algorithm is Punctured Elias Code is more efficient in terms of Space Saving compared to the algorithm Yamamoto Recursive Code.
Because the algorithm that has the most Space Saving values high then the algorithm is better to use for compressing files, in this case the string file is homogeneous.

![Comparison of Compression Time](image1.png)

**Figure 4.** Compression Time Graph to Homogeneous String Length.

From Figure 4 Compression Time Graph can be concluded that Punctured Elias Code algorithm takes shorter time than with the Yamamoto Recursive Code algorithm.

![Comparison of Decompression Time](image2.png)

**Figure 5.** Decompression Time Graph to String Length.

From Figure 5 Decompression Time Graph can be concluded that Punctured Elias Code algorithm takes shorter time than with the Yamamoto Recursive Code algorithm.

### 3.2 Heterogeneous String Testing

**Table 3.** The Experiment Results of Heterogeneous Strings with Yamamoto Recursive codes.

| Uncompressed (bits) | Char | Compressed (bits) | Compression Ratio | Bitrate (bits/symbol) | Space Saving | Compression time | Decompression time |
|--------------------|------|-------------------|------------------|-----------------------|--------------|-----------------|-------------------|
| 508.947            | 77   | 508.947           | 0.87             | 5748                  | 13%          | 100 ms          | 2990 ms           |
| 1.000.005          | 69   | 885.960           | 0.89             | 12840                 | 11%          | 199 ms          | 5364 ms           |
| 2.000.007          | 70   | 1.773.011         | 0.89             | 25328                 | 11%          | 375 ms          | 10849 ms          |
| 4.862.971          | 70   | 4.332.774         | 0.89             | 61896                 | 11%          | 943 ms          | 26382 ms          |
### Table 4. The Experiment Results of Heterogeneous Strings with Punctured Elias Code.

| Uncompressed (bits) | Char | Compressed (bits) | Compression Ratio | Bitrate (bits/symbol) | Space Saving | Compression time | Decompression |
|---------------------|------|-------------------|-------------------|------------------------|--------------|------------------|---------------|
| 508.947             | 77   | 348.839           | 0.69              | 4530                   | 31%          | 112 ms           | 2228 ms       |
| 1.000.005           | 69   | 700.242           | 0.70              | 10148                  | 30%          | 197 ms           | 5893 ms       |
| 2.000.007           | 70   | 1.410.526         | 0.70              | 20150                  | 30%          | 349 ms           | 11552 ms      |
| 4.862.971           | 70   | 3.444.838         | 0.71              | 49211                  | 29%          | 918 ms           | 28165 ms      |

From the experiment, the results of Table 3 and Table 4 it can make a comparison chart of the results Heterogeneous String testing Yamamoto Recursive Code algorithm and algorithm Punctured Elias Code based on the variable Bitrate, Compression Ratio (Cr), Space Saving, Compression Time as in Figure 6, Figure 7, Figure 8, Figure 9 and Figure 10.

![Comparison BitRate](image)

**Figure 6.** Bitrate Graph to Heterogeneous String Length.

From Figure 6 the Bitrate Graph can be concluded that the algorithm Punctured Elias Code is more efficient in terms of bitrate compared to Yamamoto Recursive Code algorithm. Due to the value of the Bitrate owned by Punctured Elias Code algorithm is lower.

![Comparison of Compression Ratio](image)

**Figure 7.** Graph of Compression ratio to Heterogeneous String Length.

From Figure 7 Compression ratio Graph it can be concluded that the Punctured Elias algorithm Code is more efficient in terms of Compression Ratio compared to the algorithm Yamamoto Recursive Code. Because the algorithm that has the most Compression ratio value low then the algorithm is better used to compress files, in this case the string file is heterogeneous.
Figure 8. Space Saving Graph to Heterogeneous String Length.

From Figure 8 the Space Saving graph can be concluded that the Punctured Elias algorithm Code is more efficient in terms of Space Saving compared to Yamamoto Recursive Code algorithm. Because the algorithm that has the most Space Saving values high then the algorithm is better to use for compressing files, in this case the string file is heterogeneous.

Figure 9. Compression Time Graph to Heterogeneous String Length.

From Figure 9 Compression Time Graph can be concluded that the algorithm Punctured Elias Code requires a shorter time compared to Yamamoto Recursive Code algorithm on length strings <1,000,000 but on strings that have length > 1,000,000 algorithm Yamamoto Recursive Code requires a shorter time.

Figure 10. Decompression Time Graph to Heterogeneous String Length.

From Figure 10 Decompression Time Graph can be concluded that the algorithm Punctured Elias Code requires a shorter time compared to Yamamoto Recursive Code algorithm on length strings
<1,000,000 but on strings that have length> 1,000,000 algorithm Yamamoto Recursive Code requires a shorter time.

Based on Table 1, 2, 3 and 4 it can be noted that in most of the cases, the Punctured Elias codes have bigger compression ratio and space savings than the Yamamoto Recursive codes. The overall bitrates of the Punctured Elias codes are also lower than those of the Yamamoto Recursive codes. Thus, it is reasonable to conclude that in general, the Punctured codes are more efficient than the Yamamoto Recursive codes in compressing texts.

4. Conclusion
The conclusion of this research are based on the experiment results on this text file compression application, on testing same character (homogeneous) and different character (heterogeneous) based on the Compression ratio (Cr) of the Punctured Elias Code algorithm better than the average Yamamoto Recursive Code algorithm Compression Ratio of 0.13 and 0.70, and the average of Bit rate 187502 and 16807, the average Space Of Saving 87% dan 30%, it means that in the vast majority of cases of text compression, the Punctured Elias codes clearly outperform the Goldbach codes in terms of compression ratio, the space savings, and bitrate.

5. Acknowledgments
The authors gratefully thank to all who have supported in the completion of this research. And to the Universitas Sumatera Utara that has given me the opportunity to publish this paper.

References
[1] David, S. Dan Giovanni M. (2010). Handbook of Data Compression. Fifth Edition, Computer Science Departement California State University Northridge Spinger-Verlag London. Spinger.
[2] Fenwick, Peter., 1996. Punctured Elias Codes for variable-length coding of the integers. Department of Computer Science. The University of Auckland, Auckland, New Zealand.
[3] Rahmat R F, Andreas T S M, Fahmi F, Pasha M F, Alzahrani M Y, Budiarto R, 2019. Analysis of dicom image compression alternative using huffman coding. Journal of Healthcare Engineering 2019, 5810540.
[4] Salomon, David (2007). Variabel-Length Codes for Data Compression. Computer Science Departement California State University Northridge USA. Spinger.