Performance Analysis of Start-Step-Stop Codes and Gopala-Hemachandra Codes2 (GH\textsubscript{2}(n)) As compression Algorithms on Text Files

M A Budiman\textsuperscript{1}, Herriyance\textsuperscript{2}, and Khairunnada\textsuperscript{3}

\textsuperscript{1}Departemen Ilmu Komputer, Fakultas Ilmu Komputer dan Teknologi Informasi, Universitas Sumatera Utara, Jl. Universitas No. 9-A, Kampus USU, Medan 20155, Indonesia
Email: mandrib@usu.ac.id, herriyance_usu@yahoo.com, khairunnadaarief@yahoo.co.id

Abstract. Data compression was born as a solution to human needs for digital data. This research compares two compression algorithms, namely the Start-Step-Stop code algorithm and Gopala-Hemachandra codes 2 (GH\textsubscript{2}(n)) algorithm in the case of text file compression. compression ratio (CR), space savings (SS), bit rate and running time are calculated in the test as consideration to compare the advantages of both algorithms. Tests carried out using homogeneous strings, heterogeneous strings and artificial corpus. In homogeneous string compression, both algorithms have the same advantages because the value of CR, SS and bit rate are the same for each number of strings used. From the time of compression, Start-Step-Stop algorithm faster than the Gopala-Hemachandra algorithm. Testing of heterogeneous strings found that the Start-Step-Stop algorithm is superior to the Gopala-Hemachandra algorithm because it can save more storage space.

1. Introduction

Data compression is a science to compress data while maintaining the original data structure in order to reduce the number of bits in the data [6]. In the process of data compression there are two components that are interrelated with each other, the process includes encoding and decoding. In the encoding process a compression representation is produced, then the decoding process plays a role in reconstructing data based on compression representations [3].

Compression algorithms are divided into two categories. First, lossless compression is compression that is done without involving loss of information so that compressed data can be recovered. Second, lossy compression is compression which results in losing some information so that it cannot be properly recovered [6]. This research discusses two lossless compression algorithms, namely Start-Step-Stop codes and Gopala-Hemachandra codes.

The Start-Step-Stop algorithm is proposed by Fiala and Greene, where coding is based on calculating the start, step, and stop values as parameter triplets used in non negative integers [5].

The Gopala-Hemachandra codes is a variation of the Fibonacci sequence series [7]. This algorithm represents a positive integer uniquely by adopting the concept of Zeckendorf's theorem as well as on the Fibonacci code to produce codes that it can represent the encode of a character. Even with the same concept, the GH code is longer than the Fibonacci codes [7]. In this study the variation of the GH code used was Gopala-Hemachandra codes 2 (GH\textsubscript{2}(n)).
The purpose of this research is to find out which algorithm is more efficient between the Start-Step-Stop algorithm and the Gopala-Hemachandra algorithm in compressing text files based on test parameters including compression file size, compression ratio (CR), space savings (SS), and bit rate; these parameters have been used to study the efficiency of Goldbach G0 codes and Even-Rodeh codes (see [8]). In addition, compression time and decompression time are also measured.

2. Method

2.1. Start-Step-Stop Algorithm

The steps to generate a Start-Step-Stop code are as follows:

- Determine the value of \( n = 0 \).
- Determine the value of \( a = \text{start} + n \times \text{step} \).
- Each subset that starts with \( n \) will be preceded by a number '1' and is followed by an insert of '0', followed by a combination of bits \( a \). So the value is \( 2a \).
- Add the \( n \) value to the next step.
  - If \( n < \text{stop} \), then go to step 2.
  - If \( n > \text{stop} \), an error message will appear and the program will stop.
  - If \( n = \text{stop} \), repeat steps 2 and 3 without inserting 0 bits from step 3.

| \( n \) | \( a = 3 + n \times 2 \) | nth codeword | Number of codeword | Range of Integers |
|---|---|---|---|---|
| 0 | 3 | 0xxx | \( 2^1 = 8 \) | 0 - 7 |
| 1 | 5 | 10xxxx | \( 2^5 = 32 \) | 8 - 39 |
| 2 | 7 | 11xxxxxxx | \( 2^7 = 128 \) | 40 - 167 |
| 3 | 9 | 111xxxxxxx | \( 2^9 = 512 \) | 168 - 679 |
| | | Total | 680 | |

2.2. Gopala-Hemachandra Algorithm

An integer can be written into the Zeckendorf representation, which is the number of non-contiguous Fibonacci numbers [7]. The Gopala-Hemachandra code is not available for positive integers and is only available for \(-20 \leq k \leq -2\) [4]. The second order Fibonacci code variation defines the Gopala-Hemachandra Code sequence with a negative \( a \) value and produces a value of \( b = 1 - a \) which can be formulated as follows [2]:

\[
GH(0) = a
\]

\((a \in Z)\)

\[
GH(1) = 1 - a
\]

and for \( n \geq 2 \),

\[
GH(n) = GH(n - 1) + GH(n - 2)
\]

Produce:

\[
GH(k) = \{a, 1 - a, 1, 2 - a, 3 - a, 5 - 2a, 8 - 3a, 13 - 5a, \ldots\}
\]
Table 2. Variation of Gopala Hemachandra Sequence [1]

| K | GH₂(n) | GH₃(n) | GH₄(n) | GH₅(n) |
|---|---|---|---|---|
| 1 | -2 | 3 | 1 | 4 |
| 2 | -3 | 4 | 1 | 5 |
| 3 | -4 | 5 | 1 | 6 |
| 4 | -5 | 6 | 1 | 7 |
| 5 | -6 | 7 | 1 | 8 |
| 6 | -7 | 8 | 1 | 9 |
| 7 | -8 | 9 | 1 | 10 |
| 8 | -9 | 10 | 1 | 11 |
| 9 | -10 | 11 | 1 | 12 |

The steps to generate the Gopala-Hemachandra code are as follows:
1. In accordance with Zeckendorf's theorem,
   \[ n = GH(i_1) + GH(i_2) + \cdots + GH(i_p) \]
with:
   GH(i₁) = the largest number of Gopala-Hemachandra which is less than or equal to \( n \)
   GH(i₂) = the largest Gopala-Hemachandra number that is less than or equal to \( n - F(i_1) \)
   So on.
2. Place the number 1 in position \( i_1, i_2, \ldots, i_p \), while the remaining positions are all filled with zeros.
3. Add number 1 at the end, so the code ends with "11". That way, according to Zeckendorf's theorem there are no 1 consecutive numbers before the end of the code.

Table 3 Various of Gopala-Hemachandra Code [7]

| N | GH₂(n) | GH₃(n) | GH₄(n) | GH₅(n) |
|---|---|---|---|---|
| 1 | 0011 | 0011 | 0011 | 0011 |
| 2 | 10011 | 10011 | 10011 | 10011 |
| 3 | 100011 | 100011 | 100011 | 100011 |
| 4 | 00011 | 011 | 101011 | 101011 |
| 5 | 000011 | 0011 | 011 | N/A |
| 6 | 001011 | 00011 | 00011 | 011 |
| 7 | 100011 | 01011 | 00011 | 0011 |
| 8 | 010011 | 100011 | 001101 | 00011 |
| 9 | 000011 | 01011 | 100011 | 001011 |
| 10 | 0010011 | 01011 | 1010011 | 100011 |
| 11 | 1001011 | 0000111 | 01011 | 1010011 |
| 12 | 0000011 | 0010011 | 010011 | N/A |
| 13 | 0001011 | 1001011 | 0000111 | 01011 |
| 14 | 00000011 | 10000011 | 0010011 | 010011 |
| 15 | 00100011 | 0100011 | 1001011 | 0000111 |

3. Results and Discussion
In this research, testing was carried out on homogeneous strings, heterogeneous strings and the artificial corpus with results presented in table 4 to table 9.
Table 4. The Experimental Result of Start-Step-Stop Code for Homogeneous String

| String | Un-compressed Bits | Compressed Bits | CR  | SS (%) | Bit rate (bits/symbol) | Compress Time (ms) | Decompress Time (ms) |
|--------|-------------------|----------------|-----|--------|------------------------|--------------------|--------------------|
| 1      | 8                 | 16             | 0.5 | -100   | 16                     | 0.001              | 0.002              |
| 10     | 80                | 48             | 1.667 | 40     | 48                     | 0.001              | 0.003              |
| 100    | 800               | 408            | 1.96 | 49     | 408                    | 0.002              | 0.003              |
| 1000   | 8000              | 40008          | 1.996 | 49.9   | 4008                   | 0.007              | 0.009              |
| 10000  | 80000             | 400008         | 1.9996 | 49.99 | 400008                 | 0.031              | 0.042              |
| 1000000| 8000000           | 40000008       | 1.99996 | 49.999 | 40000008               | 0.257              | 0.354              |

Based on the test results on homogeneous strings in Table 4 and Table 5, the Start-Step-Stop codes and Gopala-Hemachandra codes produce the same value in the compression ratio, space savings, and bit rate for each amount the string is because the code for n = 1 in the Start-Step-Stop codes and Gopala-Hemachandra Codes both amounts to 4 bits. Compressing text files containing homogeneous strings using the Start-Step-Stop codes is faster than using the Gopala-Hemachandra codes.

Table 5. The Experimental Result of Gopala-Hemachandra Code for Homogeneous String

| String | Un-compressed Bits | Compressed Bits | CR  | SS (%) | Bit rate (bits/symbol) | Compress Time (ms) | Decompress Time (ms) |
|--------|-------------------|----------------|-----|--------|------------------------|--------------------|--------------------|
| 1      | 8                 | 16             | 0.5 | -100   | 16                     | 0.004              | 0.002              |
| 10     | 80                | 48             | 1.667 | 40     | 48                     | 0.004              | 0.002              |
| 100    | 800               | 408            | 1.96 | 49     | 408                    | 0.005              | 0.003              |
| 1000   | 8000              | 40008          | 1.996 | 49.9   | 4008                   | 0.005              | 0.011              |
| 10000  | 80000             | 400008         | 1.9996 | 49.99 | 400008                 | 0.033              | 0.044              |
| 1000000| 8000000           | 40000008       | 1.99996 | 49.999| 40000008               | 0.296              | 0.363              |
| 10000000| 80000000         | 400000008      | 1.999996 | 49.9999| 400000008              | 2.999               | 3.497              |

Table 6. The Experimental Result of Start-Step-Stop Code for Heterogeneous String

| String | Char | Un-compressed Bits | Compressed Bits | CR  | SS (%) | Bit rate (bits/symbol) | Compress Time (ms) | Decompress Time (ms) |
|--------|------|-------------------|----------------|-----|--------|------------------------|--------------------|--------------------|
| 1      | 1    | 8                 | 16             | 0.5 | -100   | 16                     | 0.001              | 0.002              |
| 10     | 10   | 80                | 56             | 1.4285 | 30     | 5.6                     | 0.002              | 0.002              |
| 100    | 100  | 800               | 472            | 1.6949 | 41     | 47.2                    | 0.005              | 0.003              |
| 1000   | 1000 | 8000              | 4608           | 1.7361 | 42.4   | 460.8                   | 0.008              | 0.014              |
| 10000  | 10000| 80000             | 46008          | 1.7388 | 42.49  | 4600.8                  | 0.042              | 0.053              |
| 1000000| 100000| 8000000        | 4600008        | 1.7391 | 42.499 | 460000.8                | 0.362              | 0.419              |
| 10000000| 10000000| 800000000  | 460000008      | 1.7391 | 42.4999| 4600000.8               | 3.862              | 4.286              |

4
Table 7. The Experimental Result of Gopala-Hemachandra Code for Heterogeneous String

| String | Char | Un-compressed Bits | Compressed Bits | \( C_R \) | SS (%) | Bit rate (bits/symbol) | Compress Time (ms) | Decompress Time (ms) |
|--------|------|--------------------|----------------|----------|--------|------------------------|-------------------|--------------------|
| 1      | 8    | 16                 | 0.5            | -100     | 16     | 0.004                  | 0.002             |
| 10     | 80   | 72                 | 1.1111         | 10       | 7.2    | 0.004                  | 0.002             |
| 100    | 800  | 600                | 1.3333         | 25       | 60     | 0.005                  | 0.004             |
| 1000   | 8000 | 5912               | 1.3531         | 26.1     | 591.2  | 0.011                  | 0.009             |
| 10000  | 80000| 59008              | 1.3557         | 26.24    | 5900.8 | 0.049                  | 0.052             |
| 100000 | 800000| 590008             | 1.3559        | 26.249   | 59000.8| 0.498                  | 0.551             |
| 1000000| 8000000| 5900008            | 1.3559       | 26.2499  | 590000.8| 4.869                 | 4.397             |

Based on the test results of heterogeneous strings in table 6 and table 7, the Start-Step-Stop Code obtained the value of compression ratio and space savings greater than the Gopala-Hemachandra Code. In addition, a bit rate value that is smaller than Gopala-Hemachandra Code is obtained. From the whole test it can be seen that the Start-Step-Stop Code is proven to be faster than the Gopala-Hemachandra Code.

Table 8. The Experimental Result of Start-Step-Stop Code for Artificial Corpus

| File       | Un-compressed Bits | Compressed Bits | \( C_R \) | SS (%) | Bit rate (bits/symbol) | Compress Time (ms) | Decompress Time (ms) |
|------------|--------------------|----------------|----------|--------|------------------------|-------------------|--------------------|
| a.txt      | 8                  | 16             | 0.5      | -100   | 16                     | 0.004             | 0.002             |
| aaa.txt    | 80000              | 400008         | 1.7391   | 49.999 | 400008                 | 0.249             | 0.352             |
| alphabet.txt | 800000          | 607696         | 1.3164   | 24.038 | 23372.9                | 0.663             | 0.596             |
| random.txt | 800000             | 770392         | 1.0384   | 3.701  | 12037.3                | 0.722             | 1.002             |

Table 9. The Experimental Result of Gopala-Hemachandra Code for Artificial Corpus

| File       | Un-compressed Bits | Compressed Bits | \( C_R \) | SS (%) | Bit rate (bits/symbol) | Compress Time (ms) | Decompress Time (ms) |
|------------|--------------------|----------------|----------|--------|------------------------|-------------------|--------------------|
| a.txt      | 8                  | 16             | 0.5      | -100   | 16                     | 0.004             | 0.002             |
| aaa.txt    | 80000              | 400008         | 1.3559   | 49.999 | 400008                 | 0.299             | 0.366             |
| alphabet.txt | 800000          | 730768         | 1.0947   | 8.6539 | 28106.4                | 0.607             | 0.589             |
| random.txt | 800000             | 881912         | 0.9071   | -10.239| 13779.8                | 0.821             | 0.968             |

Based on the test results on the Artificial Corpus in Table 8 and Table 9, the Start-Step-Stop Code obtained the value of compression ratio and space savings greater than Gopala-Hemachandra Code. In addition, a bit rate value that is smaller than Gopala-Hemachandra Code is obtained. From the whole test it can be seen that the Start-Step-Stop Code is proven to be faster than the Gopala-Hemachandra Code.

4. Conclusion
From this research, it can be concluded that the Start-Step-Stop Code algorithm is better than the Gopala-Hemachandra Code 2 \((\text{GH}_2(n))\) algorithm based on the test parameters of compression ratio, space savings, bit rate and running time.
5. Acknowledgments
We are very grateful for the cooperation of colleagues at the Departemen Ilmu Komputer, Fakultas Ilmu Komputer dan Teknologi Informasi, Universitas Sumatera Utara.

References
[1] Basu M and Das M 2016 Uses of second order variant Fibonacci universal code in cryptography Control and Cybernetics 45
[2] Basu M and Prasad B 2010 Long range variations on the Fibonacci universal code. Journal of Number Theory 130 9 1925-1931
[3] Blelloch G E 2013 Introduction to Data Compression. United States: Carnegie Mellon University
[4] Nalli A and Ozyilmaz C 2015. The third order variations on the Fibonacci universal code. J. of Number Theory 149 15-32.
[5] Salomon D 2007 Variable-length codes for data compression. Springer Science & Business Media.
[6] Sayood K 2012 Introduction to Data Compression Fourth Edition. San Fransisco: Morgan Kaufman Publishers.
[7] Thomas J H 2007 Variations on the Fibonacci universal code. arXiv preprint cs/070108
[8] Budiman M A and Rachmawati D 2017 On Using Goldbach G0 Codes and Even-Rodeh Codes for Text Compression on Using Goldbach G0 Codes and Even-Rodeh Codes for Text Compression. IOP Conference Series: Materials Science and Engineering 180(1), p. 012062. IOP Publishing.