DYNAMIC UNBOUNDED INTEGER COMPRESSION

by

Naga Sai Gowtham Mulpuri, B.S.

A thesis submitted to the Graduate Council of
Texas State University in partial fulfillment
of the requirements for the degree of
Master of Science
with a Major in Computer Science
August 2016

Committee Members:

Dan Tamir, Chair

Byron Gao

Martin Burtscher

FAIR USE AND AUTHOR'S PERMISSION STATEMENT

Fair Use

Duplication Permission

DEDICATION

I would like to dedicate this thesis to my wife, Keerthi Mulpuri.

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to my advisor, Dr. Dan Tamir, for his continuous support throughout my research. I am glad that I worked under a professor who understood me in my odd times and whose guidance helped me in all the time of research and writing this thesis. I specially thank Dr. Dan Tamir for everything he taught me during the last year.

In addition, I would like to thank Dr. Byron Gao and Dr. Martin Burtscher for serving in my committee and supplying valuable feedback.

TABLE OF CONTENTS

Page

LIST OF FIGURES

Figure **Page**

ABSTRACT

One of the important applications of data compression is inverted indexes compression, which is a lossless compression. Presently, most of the internet search engines such as Google and Wikipedia process a large amount of documents and employ information retrieval policies through inverted indexes for fast search or query. This might necessitate compressing the data in order to store it in physical memory and enable computationally effective retrieval of information. Nevertheless, their currently used compression methods implement static compression procedures and do not fully exploit dynamic integer compression capabilities.

In this research, we have exploited dynamic Huffman compression method referred to as δ-Huffman coding for compressing integers and improved the attainable compression ratios. To achieve this, we have explored a combination of static and dynamic compression algorithms. To the best of our knowledge this is the first work that combines unbounded integer compression methods with Huffman coding and comparatively evaluates their performance.

We have applied compression algorithms on the data-sets drawn from Geometric probability distribution function, Poisson distributions, realistic inverted lists, and page-rank lists obtained from Wikipedia. Our results show that the bit-rate of δ-Huffman coding is the best of all of the tested coding methods including Comma code, Elias Gamma code, Elias Delta code, Elias Omega code, Fibonacci code, and Golomb code. Additionally, it is very close to the estimated entropy of the test sequences.

CHAPTER 1

INTRODUCTION

Storing data in an effective and compact form is one of the most challenging problems in present times. Following the formulation of Information theory by Shannon [2], Huffman introduced a method for lossless compression of information sources that generate finite length strings from a finite alphabet with known probability distribution [3]. During the same time, Elias introduced methods for compression of unbounded integers under a relatively weak assumption about their probability of occurrence or probability distribution function (PDF) [6]. Namely, his assumption is that the PDF of the integers is a monotonically decreasing function of these integers. The above stated two methods (Huffman and Elias – Gamma, Delta and Omega) are static compression methods [4].

Later on, the field of lossless data compression underwent tremendous amount of theoretical and practical advances including the introduction of numerous static algorithms for finite length strings and unbounded integers [5, 7, 11], as well as dynamic algorithms for the compression of finite length strings [5, 11].

Recent developments in computer science include fast web search and information retrieval (IR) on a massive scale. In UNIX the "grep" command had initially been used to search through text documents, but currently search policies are executed over billions of documents and terabytes of data. This makes the IR process challenging. Nevertheless, inverted indexes can be used to achieve fast and efficient IR [1, 5, 10]. Presently, most of the internet search engines like Google and Wikipedia having a huge amount of documents (or data) employ IR policies through inverted indexes for fast search or query. This might necessitate compressing the data in order to store it in physical memory and enable

computationally effective retrieval of information. Compression methods, mainly in the form of integer compression, are currently employed for inverted list compression. Nevertheless, these methods use static compression procedures and do not fully exploit dynamic integer compression capabilities.

The problem addressed in this thesis is devising methods for effective retrieval of information via data compression. The proposed solution is to explore the possibility of combining integer compression methods with a new dynamic Huffman compression algorithm referred to as δ-Huffman coding for compressing integers. To this end, we have conducted a set of experiments for analyzing and comparing the performance of static compression methods Comma code, Elias Gamma code, Elias Delta code, Elias Omega code, Fibonacci code, and Golomb code as well as δ-Huffman coding (see section 2 for definitions) on both synthetic data and real inverted list data [19, 20]. Additionally, we have compared the bit-rate of δ-Huffman to the bit-rate of the compression techniques "gzip" and "bzip2" (see section 2 for definitions).

Note that in this research, we are not concerned with the decompression, throughput, procedure, and latency of the algorithms. The main goal of this research is to explore ways for achieving high compression ratio (or communication rate which is measured as the average of total number of the bits/integer which is called as the bit-rate in this thesis).

The hypothesis of this thesis is that the δ-Huffman code can be implemented on inverted lists and provide low bit-rate that is very close to the entropy of the data [2].

The main contribution of this study is the exploration of achievable compression ratios in the context of δ-Huffman encoding. To the best of our knowledge this is the first work that combines unbounded integer compression methods with Huffman coding and comparatively

evaluates their performance.

The rest of this thesis is organized as follows: Chapter 2 provides background material. Chapter 3 includes literature review. Chapter 4 details the research methodology and the experimental procedures. Chapter 5 analyses the results and Chapter 6 includes the conclusion and future work.

CHAPTER 2

BACKGROUND

In this chapter, we explain the relevant terms related to data compression, integer compression, and IR which are implemented in this research.

2.1 Data Compression

Data compression is concerned with reducing the number of bits needed to store or transmit data. Bit-level compression is generating a variable number of bits for each input symbol. Byte-level compression is encoding symbols' bits and/or bytes into a variable number of bytes. One of the important applications of data compression is in the field of IR.

2.1.1 Entropy

Suppose that the alphabet of a source X is $A_x = \{a_1, \dots, a_n\}$ with probabilities $(p_1 \dots p_n)$ then the entropy ($H(X)$) is given by, $\sum_1^n p_i \, log_2 \left(\frac{1}{p_i}\right)$. The entropy rate of a data source is a lower bound on the average number of bits per symbol / integer needed to encode it.

2.1.2 Prefix Codes and Universal Codes

A code is a mapping and it maps source symbols into code words. Mathematically, a source of messages is a pair (M, P) where M is possibly an infinite set of messages and P is a function that assigns a nonzero probability to each message [20]. A message is mapped into a bit string whose length depends on the quality of the compression and on the probabilities of the individual symbols. A code is universal if it compresses messages to code words whose average length is bounded by $C_1 H_1 + C_2$ where C_1 and C_2 are constants greater than or equal to 1. A code with $C_1 = 1$ is called asymptotically optimal.

2.2 Types of Compression

2.2.1 Lossy Compression

High compression ratios can be achieved with lossy compression, which discards some of the information. Lossy compression is useful when the information lost does not cause any significant distortion. In most cases of IR, one cannot afford losing information.

2.2.2 Lossless Compression

Lossless compression is a class of data compression algorithms where the original data can be perfectly reconstructed from the compressed data. Nevertheless, the compression ratio might be smaller than that of lossy compression. A code is uniquely decodable (UD) if it is a prefix code [21, 22]. Integer compression is a special case of lossless compression.

2.2.3 Byte-level Compression and Bit-level Compression

Compression of data into a variable number of bits is considered bit-level compression. Compression of data into a variable number of bytes is considered as byte-level compression. Several byte-level coding techniques exist. In one of the variants of the byte-level codes, a non-negative integer n is encoded as a sequence of bytes. In each byte the lower 7 bits are used to code the magnitude of n and the most significant bit is reserved as a flag bit indicating whether the next byte is still a part of the current magnitude [6, 17]. All of the compression methods used in this thesis are bit-level compression techniques.

2.3 Integer Compression Algorithms

Several integer compression methods employ lossless compression on integers under the assumption that the probability of occurrence of "small integers" is larger than the probability of the occurrence of "large integers" [2]. That is, $p(n) \geq p(n+1)$. These techniques assign variable length code (VLC) to integers allocating more bits to larger integers. Nevertheless, using VLC requires special provisions to ensure that the code is UD [2, 3, 6].

Numerous lossless compression methods, including Huffman Compression, dictionary based LZ compression, and arithmetic coding exist [3]. These methods are generally not suitable for integer compression and assume finite alphabet with known probabilities. For example, static Huffman coding requires knowledge of the precise distribution of integers beforehand and dynamic Huffman coding cannot readily handle unbounded integers.

There are numerous integer compression methods, including Comma code, Elias Gamma code, Elias Delta code, Elias Omega code, Fibonacci code, Golomb code, Unary code, and Variable Byte code. These methods are reviewed in the next paragraphs.

2.3.1 Unary Code

In Unary code a positive integer n is encoded as n bits of '1' ('0') followed by a single bit of '0' (1). Hence the 0-bit (1-bit) serves as a comma [5]. This code is not universal and may have high compression ratio only under a restrictive set of assumptions about the input integers probability distribution function (PDF).

2.3.2 Fibonacci Code

The Fibonacci code exploits the fact that each integer has a unique representation as a sum of nonconsecutive elements of the Fibonacci sequence. It is a universal code. For any positive integer n, it can be represented as $n = b_1F_1 + b_2F_2 + \cdots \cdot b_nF_n$ where F_n is the n^{th} Fibonacci number and 'b' is either $0\ or\ 1$. This is called Fibonacci representation of integer.

For example, $33 = 1 + 3 + 8 + 21$, which is expressed as 1010101. The Fibonacci representation has important property, that the representation does not contain any adjacent $1's$ since …01100… can be represented as …00010… from the definition of Fibonacci numbers,$F = F_{n-1} + F_{n-2}$. Fibonacci code of the positive integer is the Fibonacci

representation of n with additional 1 appended to right end. For example, the Fibonacci code of $33\ is\ 1010101|1, and\ that\ of\ 5\ is\ 0001|1$. Since Fibonacci representation does not contain adjacent $1's$, it's obvious that a pair 11 means the end of the code, which makes the code uniquely decodable.

2.3.3 Elias Gamma Code

In Elias Gamma code, a non-negative integer n is represented by the length part and the data part [7]. The data part ($B(n)$) is the binary representation of n. The length part ($|B(n)| - 1$), where $|B(n)|$ is the number of bits in $B(n)$, is encoded using $|B(n)| - 1$ zeros. It provides the number of bits (minus 1) used in the binary representation of n.

Elias Gamma code can be used to encode positive unbounded integers. Elias Gamma code provides high compression when the data-set is having small integers and rare large integers. Elias Gamma code is a universal code.

2.3.4 Elias Delta Code

An integer n is represented by encoding the $|B(n)| - 1$ field of the Elias Gamma using Gamma code, followed by $B(n)$ without its most significant bit. Elias Delta code is relatively efficient and it is asymptotically optimal.

Example: For N = 17, its binary representation is $10001\ (five\ bits)$. Remove the leftmost 1 and prepend $5 = 101$ yields 1010001 and number of newly added bits is 3. Prepending $2\ zeros$ to the code makes it 001010001. This is the Elias Delta code for 17.

2.3.5 Elias Omega Code

Elias Omega code apply recursion by employing Gamma Code to the element $|B\ (n)|$.

2.3.6 Golomb Code

Golomb code (GC) for a non-negative integer n depends on the choice of a parameter (divisor) d. It encodes the integer into two parts. If the parameter is a power of 2 then the code is referred to as Rice Code. According to the Golomb algorithm, the integer part of n/d (q) is stored as unary code using $q + 1$ bits and the remainder part r is coded using efficient binary code [7]. If the parameter is not a power of 2 use, then r is can be represented by using variable length bits. Let b=$\lceil log_2 d \rceil$. If $r < 2^b - d$, then we use $b - 1$ bits to encode r, else we use $b\ bits$ to encode r.

For example, $d = 3$, N = 24 is represented by 1111111011. When $d = 3$, it produces the possible remainders 0, 1, 2 which are coded as 0, 10, 11. For N = 10, the quotient (24 – 1) / 3 = 7 is coded as unary code (11111110) and the remainder 3 is coded as binary (11). Hence GC with parameter d = 3 for N = 24 is 1111111011.

2.3.7 Ternary Comma Code

Binary ($base\ 2$) numbers are based on the two bits $0\ and\ 1$. Similarly, ternary ($base\ 3$) numbers are based on the three digits ($trits$) $0, 1\ and\ 2$. Each $trit$ can be encoded in two bits, but two bits can have four values. In ternary number system where each $trit$ is represented by two bits and in addition to the three trits there is a fourth symbol, a $comma$ implemented as "11" [19]. The comma code of n is simply the ternary representation of $n - 1$ followed by $comma$.

2.3.8 GZIP

GZIP is a compression utility which implements a variant of the LZ77 [21]. The GZIP

algorithm looks for repeating strings of length up to 256 bytes within a 32kB sliding window. It uses two Huffman trees, the first tree compresses the distances in the sliding window and the second one is used to compress the length of strings. The second tree is also used to compress individual bytes that were not part of any sequence.

2.3.9 BZIP2

BZIP2 is a lossless compression utility which implements the block sorting algorithm described by Burrows and Wheeler [22]. It compresses data in blocks, where the block size is adjustable. It uses the Burrows-Wheeler transform to convert frequently occurring character sequences into strings and then uses a move-to-front transform and Huffman coding for compressing the data. Before providing the output for a block BZIP2 performs several operations on the block to improve the compression ratio.

2.3.10 Evaluation of Compression Quality

For evaluating the quality of compression algorithms we have compared the averages of total number of bits (bit-rate) to the estimated entropy of the input data. Compression algorithms having the bit-rate close to the entropy are considered as the best compressor providing the best bit-rate. We have also applied the compression techniques "gzip" and "bzip2" on the datasets of GPDF (0.5, 0.1, 0.01), PD, and gaps of sorted inverted lists and compared its bit-rate to the compression algorithms. To compare the compression techniques "gzip" and "bzip2" to the compression techniques implemented in this research, we have converted the total bytes (of the compressed data files compressed by "gzip" and "bzip2") to the bit-rate as below:

$$bit-rate = Averages\ number\ of\ bits = \frac{\text{total number of output\_bits}}{\text{number of integers (input)}}$$

2.3.11 Huffman Code

Huffman Code (HC) is a prefix code that requires apriori knowledge of the set of probabilities of symbols. The Huffman procedure for a given alphabet and a set of probability values can be described as follows.

1. Construct a forest by starting with each symbol in its own tree
2. Unify the two trees that have the two smallest probabilities; proceeding iteratively until a single tree is left.
3. The number of bits in the Huffman code of a symbol is the depth of that symbol in the tree, and its code is a description of the path from the root (e.g., using 0 for a left branch and 1 for a right branch).

2.3.12 Dynamic Huffman Code

In Dynamic Huffman coding, neither the encoder nor the decoder know the probabilities of occurrences of symbols in the source sequence. These probabilities are estimated during the encoding. The 'NYT' is a special node used to represent symbols / integers which are 'not yet transmitted'. It is assumed to have a weight of 0. This weight is unchanged. Upon the arrival of an un-encountered symbol (say a), a is removed from the NYT. It is added to the "already transmitted" (AT) list with a weight of 1. The Huffman code of the NYT along with the fixed length code of a are transmitted to the decoder [6]. Next, using the updated AT, the current encoder/decoder Huffman trees are efficiently updated (e.g., using the sibling property). If a is in the AT, then its code in the tree is transmitted its weight is incremented by 1 and the AT tree is efficiently updated. The update procedure can use the sibling property [21] and is designed to ensure that the encoder and the decoder obtain the same tree.

2.3.13 Delta-Huffman Code

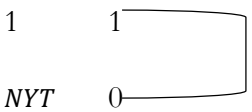
Our version of the dynamic Huffman (δ-Huffman) algorithm on unbounded integers is as follows. The 'NYT' contain symbols / integers which are 'not yet transmitted'. Upon the arrival of an un-encountered integer (say j), j is removed from the NYT. It is added to the AT list with a weight of 1. The Huffman code of the NYT along with the Elias Delta code of j are transmitted to the decoder [6]. Next, using the updated AT, the current encoder/decoder Huffman trees are efficiently updated. If j is in the AT, then its code in the tree is transmitted and its weight is incremented by 1 and the AT tree is efficiently updated. The update procedure is the same as in the dynamic Huffman algorithm.

The following example explains the algorithm of δ-Huffman encoding.

Example: Input ➔ "1, 2, 3, 2" ➔Step 0: NYT "0"

Step 1: The Input is "1", which is a new integer, send ➔ <C (NYT), δ (1))>➔ <01> [Since the Code (NYT) = "0" from previous step and δ (1) = "1"].

Now, update the tree

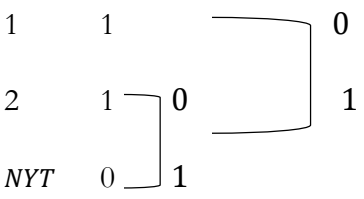


Code (1) = "0"

Code (NYT) = "1"

Step 2: Input is "2", which is a new integer, send ➔ <C (NYT), δ (2))>➔ <"10100"> [Since the Code (NYT) = "1" from previous step and δ (2) = "0100"].

Now, update the tree



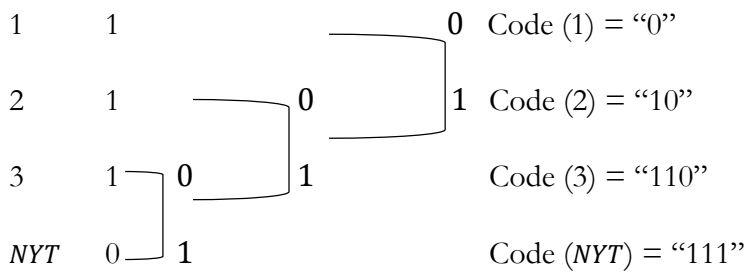
Code (1) = "0"

Code (2) = "10"

Code (NYT) = "11"

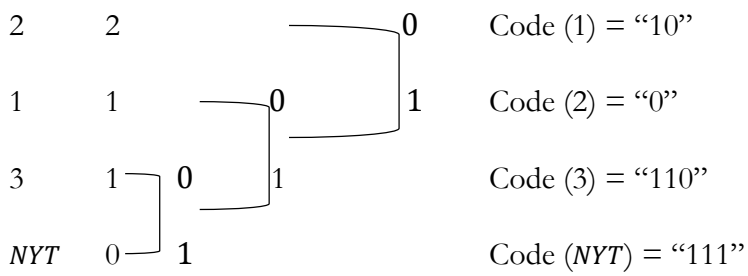
Step 3: Input is "3", which is a new integer, send ➔ <C (NYT), δ (3))>➔ <"110101"> [Since the Code (NYT) = "11" from previous step and δ (3) = "0101"].

Now, update the tree



Step 4: Input is "2", which is already on the tree, send ➔ <C (2)➔ <10> [Since, Code (2) = "10" from previous step].

Now, update the tree



The final output is:

Integer	δ-Huffman
1	01
2	10100
3	110101
2	10

2.4 Inverted Index

In this section, we provide explanation about inverted index (inverted lists) and the construction of inverted index is provided.

An inverted list is a data structure used in the context of IR for storing mappings in the form of indices, from content such as key-words or terms, to the parts of a documents where they reside. In inverted indexes, the dictionary is a data structure which consists of all the individual words or terms. The terms appearing in a document for each item in the list are called a posting. The list of all the postings is called the postings list or the inverted list. Generally, the dictionary is stored in memory with pointers to each postings list which is stored on the disk.

Within a document collection, each document is associated with a unique serial number, an integer, known as the document identifier (DocID). During the index construction, each new document is assigned with successive IDs, when it is first encountered. The D-gap is the successive difference between the document Id's in the postings list.

As the inverted list occupies a massive amount of memory for storing the dictionary file and the posting files, compression techniques can be used to reduce the space required by the inverted index thereby reducing disk and memory volume and access time.

Most current techniques for IR are using byte-level compression and in general achieve a rate that is about two times the entropy of the data [17]. We implement bit-level compression algorithm that achieves a rate that is close to the entropy of the data.

To illustrate, consider the inverted indexes as the index part of a book. The index contains key words (the set of key words can be referred to as dictionary) and the page numbers where they appear. Inverted indexes use similar concept and for each term in the dictionary the posting list contains the ID of webpages where the tem appears. In the page-

rank experiments, we have used postings list that are ordered according to the page-rank algorithm.

2.4.1 Index Construction

In order to gain the speed benefits of indexing at the retrieval, one needs to construct the index list in advance. The main steps comprise:

1. Collecting the documents to be indexed.
2. Tokenizing the text, and turning each document into a list of tokens.
3. Normalizing the tokens by linguistic preprocessing [1].
4. Indexing the document for each term that occurs by creating an inverted index list consisting of a dictionary and postings.

Generally, the next step is lexicographically sorting the list of terms. Multiple occurrences of the same term from the same document are then merged. Instances of the same term are grouped and the result is split into a dictionary and postings.

Transferring data in an uncompressed form might take more time than transferring the data in compressed form [1]. Hence, the speed and efficiency of a search engine or IR systems depends on the index construction, index compression and decompression, query processing, and the ranking algorithm.

2.5 Geometric Probability Distribution Function (GPDF)

The GPDF is the probability distribution of the number of failures in a random event before the first success, supported on the set {0, 1, 2, 3….}. It is the probability that the first occurrence of success, in a set of binary independent trials, is encountered after X trials, each with success probability p. It is given by $Probability\,(X = k) = (1-p)^{k-1}p$, for $k = 1, 2, 3, \ldots$

2.6 Poisson Distribution (PD)

Poisson distribution is a discrete probability distribution function that expresses the probability of a given number of events occurring in a fixed interval of time and/or space if these events occur with a known average rate and independently of the time since the last event. It is given by $P\ (k\ events\ in\ interval) = \frac{\lambda^k e^{-\lambda}}{k!}$ Where, λ is the average number of events per interval.

2.7 Zero Padding and Difference Sequence

Zero padding is adding zeroes to the beginning of an input stream. Subtracting successive integers produces a difference sequence (referred to as set of gaps). For example, for the input data 3, 6, 8, 10. The successive differences are 3, 3, 2, 2.

If the sequence is not sorted, we can get negative differences. However, all the methods discussed so far can only represent natural numbers. The code words for many of the methods start with "0".

In the next section, we explain the procedures for signed numbers representation.

2.8 Signed Number Representation

In several cases the difference sequence contains negative numbers (e.g., for un-sorted data). Our experimental setup has explored two types of mapping signed differences into integers.

2.8.1 Sign and Magnitude Representation

Note that in some cases the negative sign or the positive sign can be eliminated. Recall, that in many of the cases positive numbers start with "0" and there is no representation for zero. Hence:

The sign and magnitude representation to the differences (including positives and negatives) is as below:

Golomb code: positive differences ' $+\ n$' = Golomb code (n), negative differences ' $-\ n$' = 1 Golomb code $(|n| + 1|)$

Fibonacci code: positive differences ' $+\ n$' = 0 Fibonacci code (n), negative differences ' $-\ n$' = 1 Fibonacci code $(|n| + 1|)$

For Elias codes, except for the code for "1" every other code starts with "0". Hence:

γ code: positive differences ' $+\ n$' = $\gamma\ (n + 1)$, negative differences ' $-\ n$' = 1 $\gamma\ (|n| + 1|)$

δ code: positive differences ' $+\ n$' = $\delta\ (n + 1)$, negative differences ' $-\ n$' = 1 $\delta\ (|n| + 1|)$

Ω code: positive differences ' $+\ n$' = $\Omega\ (n + 1)$, negative differences ' $-\ n$' = 0 $\Omega\ (|n| + 1|)$

δ-Huffman code: positive differences ' $+\ n$' = Code (NYT) $\delta\ (n + 1)$, negative differences ' $-\ n$' = Code (NYT) 1 $\delta\ (|n| + 1|)$

Comma code: positive differences ' $+\ n$' = 0 Comma code (n), negative differences ' $-\ n$' = 1 Comma code $(|n|)$.

2.8.2 Odd-Even Mapping

We have represented signed differences of data by mapping negative integers (including '0') to odd integers and positive integers to the even integers as below:

Negative integers ' $-n' = 2 * |n| + 1$

Positive integers ' $+n' = 2 * |n|$

CHAPTER 3

LITERATURE REVIEW

In this section, we analyze and compare our methodology and techniques to the techniques implemented by different authors in the areas of integer compression algorithms. We discuss how the previous compression methods are compared to the entropy. We conclude with how, our research is different from others compression techniques and yet provide better compression rate than the other compression techniques that is one times the entropy.

Anh et al. have proposed a new schema for compressing lists of integers in IR systems. Their method uses fixed binary codes for D-gaps [9]. They have shown that the computation can be very fast, as a relatively small number of masking and shifting operations are required per code word. However, they did not exploit dynamic data reordering.

Domnic at al. have proposed the use of the Extended Golomb code (EGC) for inverted list compression [4]. They, have considered neighbors in the list and have performed data reordering. They have not considered dynamic data compression. More often, EGC is actually Elias Gamma code.

Scholer et al. have proposed using byte-aligned codes instead of bit-aligned codes for storing lists of integers [10]. With the new method, they have proved that the average query response time is halved. However, they did not exploit the variety of types of variable-byte codes, the special properties of the data, and different techniques of dynamic compression.

Burtscher has done numerous experiments with lossless compression of floating-point numbers [14, 15]. Burtscher was mainly concerned with time performance and real time

performance willing to sacrifice compression quality for complying with time constraints. In our work, we deal with integers. We are more concerned with compression quality than with compression speed.

Jenkins et al. have proposed a new solution for lossless compression of floating-point data using PForDelta algorithm [16-18]. Floating point numbers are encoded using three components: a sign bit, an exponent field, and a significand field. The authors split each 64-bit double value into upper k bytes and lower $8-k$ bytes. The values in data are classified by upper part. Each class with same upper parts forms a bin. That is, a bin is the set of values in the data-set with same upper parts. In each bin, because all elements have same upper parts, it's only needed to store lower parts of values are stored. Their main goal is the decompression speed and they only deal with fixed length data. Nevertheless, they did not work on unbounded integers and variable length integers. However, our main goal and concentration of our research is to, improve the compression ratios using δ-Huffman compression technique. PFor and PForDelta are also used for inverted index compression, yet it is a byte-level algorithm [18].

Lemire et al. have compared and analyzed different byte-level compression algorithms for fast decompression of the data [17]. The authors used several byte-level compression methods and list compression rate of twice to the entropy as a "good" result. Their trade off was the compression ratio and their main goal was to increase the decompression speed. Indeed, for achieving high speed, their technique requires special decompression hardware (SIMD). However, our goal is to improve achievable compression ratio by the exploitation of Huffman coding. We are less concerned about complexity. Our technique (δ-Huffman) is a bit-level compression and achieves compression ratio that is close to one times the entropy.

CHAPTER 4

EXPERIMENTAL SETUP, EXPERIMENTS, RESULTS, AND EVALUATION

In this chapter, we detail the list of experiments performed along with the experimental setup and provide the results, and evaluation of each experiment results.

4.1 List of Experiments

Experiment 1

In this experiment, we have used the data-set of $1\ to\ 10,000$ consecutive integers and data-set of powers of 2 from 2^1 to 2^{60}. Note that we have only used powers of 2 integers for a total input of 60 entries.

Experiment 2

In this experiment, we have used the data-sets of GPDF with probabilities of 0.5, 0.1, 0.01 and PD with $\lambda = 128$.

Experiment 3

In this experiment, we have used the data-sets of gaps of GPDF with probabilities of 0.5, 0.1, 0.01 and PD with $\lambda = 128$.

Experiment 4

In this experiment, we have used the data-sets of gaps of gaps of GPDF with probabilities of 0.5, 0.1, 0.01 and PD with $\lambda = 128$.

Experiment 5

In this experiment, we have used the data-sets of gaps of realistic inverted lists obtained from Wikipedia. We have taken the top 15 data-sets of inverted lists which appeared in the most frequently occurring words in Wikipedia in the year 2015.

The data-sets are listed below:

1. "**2015**" contains 324, 888 entries. The minimum entry of the data-set is 1 and maximum is 75, 130.
2. "**Bollywood**" contains 10, 605 entries. The minimum entry of the data-set is 1 and maximum is 474, 315.
3. "**Film**" contains 470, 269 entries. The minimum entry of the data-set is 1 and maximum is 66, 129.
4. **"Grei**" contains 49, 973 entries. The minimum entry of the data-set is 1 and maximum is 421, 836.
5. "**India**" contains 470, 269 entries. The minimum entry of the data-set is 1 and maximum is 66, 129.
6. "**Rousei**" is the smallest data-set contains 211 entries. The minimum entry of the data-set is 211 and maximum is 754.
7. "**State**" is the biggest data-set which contains more than million entries (1, 237, 789 entries). The minimum entry of the data-set is 1 and maximum is 37, 064.
8. "**Stephen**" contains 94, 033 entries. The minimum entry of the data-set is 1 and maximum is 192, 999.
9. "**Walker**" 50, 874 entries. The minimum entry of the data-set is 1 and maximum is 423, 541.
10. "**War**" contains 540, 639 entries. The minimum entry of the data-set is 1 and maximum is 183, 097.
11. "**West**" contains 540, 639 entries. The minimum entry of the data-set is 1 and maximum is 183, 097.
12. "**World**" contains 843277 entries. The minimum entry of the data-set is 1 and

maximum is 86, 200.

13. "**Facebook**" contains 25, 451 entries. The minimum entry of the data-set is 1 and maximum is 438, 117.
14. "**Iraq**" contains 32,488 entries. The minimum entry of the data-set is 1 and maximum is 422, 977.
15. "**Obama**" contains 17, 153 entries. The minimum entry of the data-set is 1 and maximum is 227, 613.

Experiment 6

In this experiment, we have used the data-sets of gaps of gaps of realistic inverted lists of the terms "2015," "Bollywood," "Film," "Grei," "India," "Rousei," "State," "Stephen," "Walker," "War," "West," and "World" obtained from Wikipedia.

Experiment 7

In this experiment, we have used the data-sets of gaps of realistic page-rank lists (see background section) of the terms "Facebook," "Grei," "Iraq," and "Obama" obtained from Wikipedia. We have compared the bit-rate of the gaps of page-rank lists to the bit-rate of the gaps of inverted lists.

Experiment 8

In this experiment, we have used the data-sets of GPDF with probabilities of 0.5, 0.1, 0.01, PD with $\lambda = 128$, and the gaps of sorted inverted lists of the terms "Bollywood," "Grei," "Rousei," "Walker," "Facebook," "Iraq," "Obama," "2015," "Film," "India," "War," "West," "World," "State," and "Stephen". We have compared the bit-rate of compression algorithms Elias Delta code, δ-Huffman code to the compression techniques "gzip" and "bzip2".

Experiment 9

In this experiment, we have compared the performance of the compression algorithms

Elias Delta code, δ-Huffman code, and Aging-Huffman code on the data-sets of GPDF (0.01) and on the gaps of sorted inverted lists of the terms "Obama," "Rousei," and "Walker."

4.2 Experimental Setup

We have used the Intel Core i5-5200U CPU @2.20 GHz with 4GB RAM on a 64-bit operating system under Windows 10 for all the experiments. For all the experiments, we have used Java SE – Version: 8 as the programming language and Eclipse Java EE IDE – Version: Mars.2 Release (4.5.2) as the development environment.

In all of the listed experiments, we have performed integer compression techniques Comma code, Elias Gamma code, Elias Delta code, Elias Omega code, Fibonacci code, Golomb code (with parameters 2, 4, 8, 16), and δ-Huffman code (except for experiments 1 and 2 which did not include δ-Huffman code). In experiment 1, we have performed Golomb coding with parameter 64. We have provided the histograms for all the experiments, except for experiments 1 and 2. From the histograms we have calculated a fitted curve and the R-Squared value of the fit. R-squared is a statistical measure of how close the data to the fitted curve. We have calculated the average, variance, and entropy (see background section) for each of the experiments (we did not calculate the trivial entropy for experiment 2.)

For experiments 2, 3, and 4 we have used the data-sets drawn from GPDF with the probabilities (0.5, 0.1, 0.01) and PD with $\lambda = 128$. The main reasons for using these probabilities with these parameters are:

1. Several papers in the literature use the data-sets of GPDF and PD with these parameters [ref].
2. The data-sets of GPDF are a good representation for the inverted lists gaps. The averages of the bit-rate in most of the gaps of the realistic data-sets are closer or equal to the averages of bit-rate of gaps of GPDF (mainly GPDF 0.01)

For experiments 5, 6 we have used the data-sets of sorted inverted lists obtained from Wikipedia. We have identified the 15 most commonly occurring words in Wikipedia searches in the year of 2015 and have used the inverted lists of these words (see background section, for the detailed explanation of inverted index).

In experiment 9, we have modified the δ-Huffman code by adding the functionality of aging. In the Huffman tree, if any integer is not repeated for k steps ($k = 100$ in our experiments), we have decremented that weight of the particular integer by 1 in each consecutive step. This method is referred to as Aging-Huffman.

The next section details the experiments with the results performed in this research.

4.3 Experiments, Results, and Evaluation

4.3.1 Experiment 1

Experiment 1.1:

In experiment 1.1, we have used the data-set of $1\ to\ 10,000$ consecutive integers. Figures 1.1(a), 1.1(b) provide the total number of bits of the Golomb coding and other classical integer compression algorithms. Figures 1.1(c), 1.1(d) provide the bit-rate of the Golomb coding and other classical integer compression algorithms.

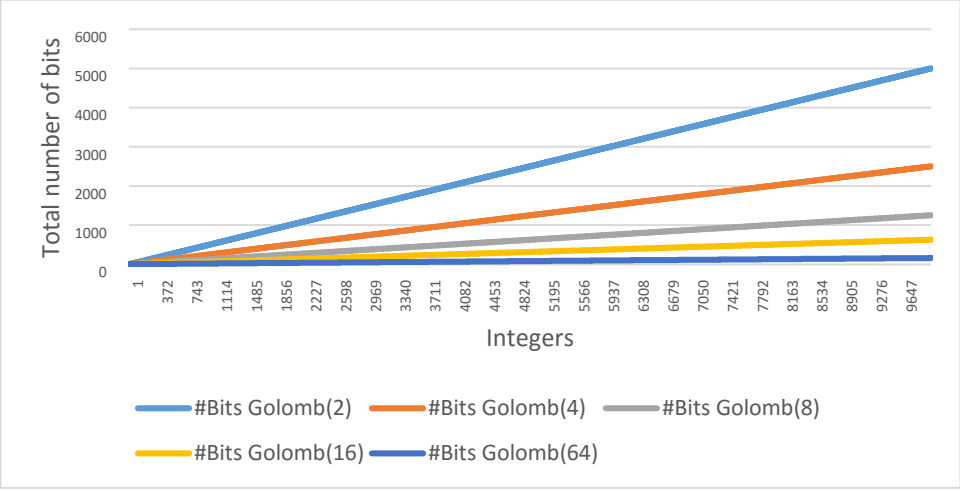


Figure 1.1(a): The total number of bits of the Golomb coding – 1-10,000

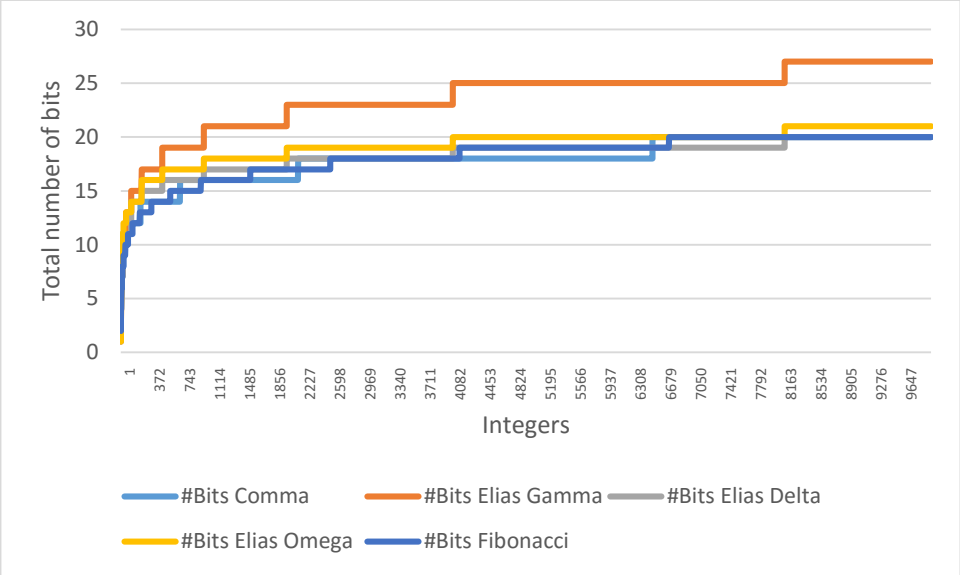


Figure 1.1(b): The total number of bits of the compression algorithms – 1-10,000

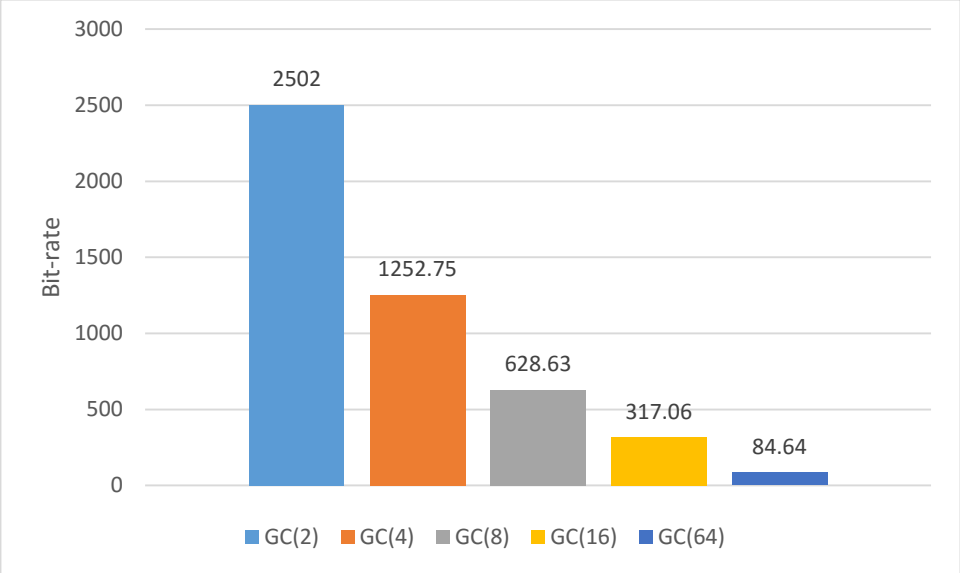


Figure 1.1(c): The bit-rate of the Golomb coding – 1-10,000

To better understand the performance of the Golomb coding with different parameters for different data-sets, we provide the bit-rate of the Golomb coding with varying parameters for GPDF (0.1).

Figure 2.2(b) provides the bit-rate of the Golomb coding for GPDF (0.1).

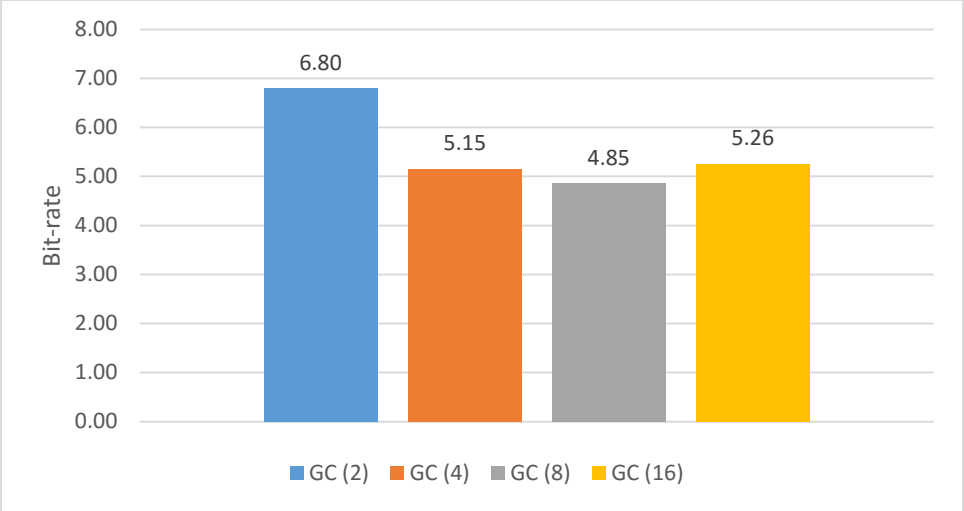


Figure 2.2(b): The bit-rate of the Golomb coding - GPDF (0.1)

Comparing figures 1.1(c) and 2.2(b), we can observe that the Golomb coding provides a low bit-rate with the high parameter (64) for the data-set of 1-10, 000 integers but in the case of GPDF (0.1), Golomb coding with the high parameter (16) provides the high bit-rate. From this, we can infer that, ideally, for effective Golomb coding prior knowledge of the data-sets should be available.

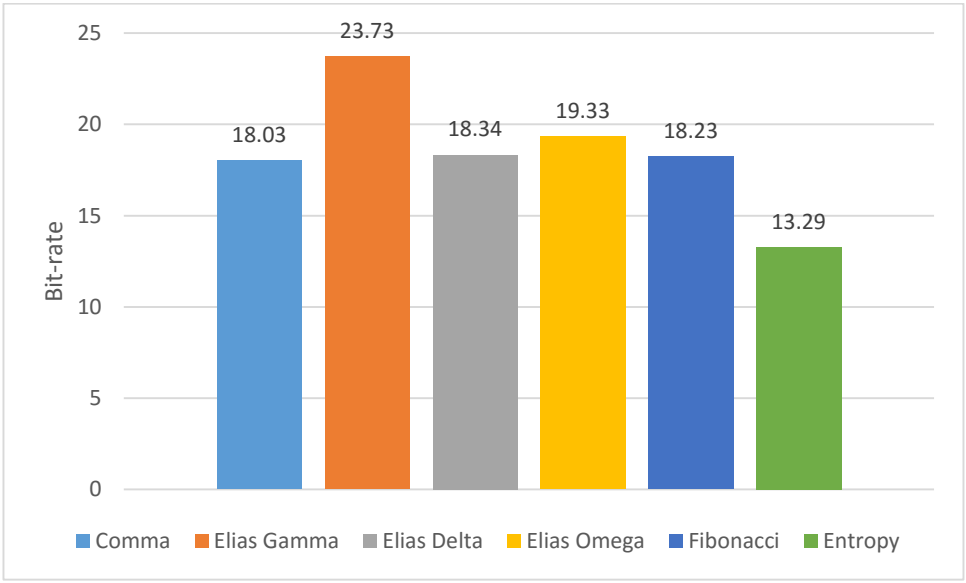


Figure 1.1(d): The bit-rate of the compression algorithms – 1-10,000

From figure 1.1(c) we can observe that among the varying parameters (2, 4, 8, 16, 64) Golomb coding with parameter 64 is achieving a low bit-rate with an average of 84.64 bits per integer. Golomb coding with parameter 2 is achieving a high bit-rate with an average of 2502 bits per integer. Using Golomb coding to provide low bit-rate without the knowledge of the data-set and the parameter is not practical in real time applications. To observe this particular behavior of the Golomb coding (with different parameters) in this research we have provided the graphs of the bit-rate of the Golomb coding (with different parameters) for every data-set.

From figure 1.1(d) we can observe that the Comma code is the best compressor in experiment 1 with an average of 18.03 bits per integer. The calculated entropy for experiment 1 data-set is 11.85 bits per integer. Fibonacci code is competitive and it is the second best compressor in this experiment with an average of 18.23 bits per integer. Elias Gamma code is the lowest performer in this experiment with an average of 23.73 bits per integer.

Experiment 1.2:

In experiment 1.2, we have used the data-set of powers of 2 from $2^1 \; to \; 2^{60}$. Note that we have used powers of 2 integers for an input of 60 entries.

Figure 1.2(a) provides the total number of bits of Comma code, Elias Gamma code, Elias Delta code, Elias Omega code, and Fibonacci code. Figure 1.2(b) provides the bit-rate of the compression algorithms such as Comma code, Elias Gamma code, Elias Delta code, Elias Omega code, and Fibonacci code.

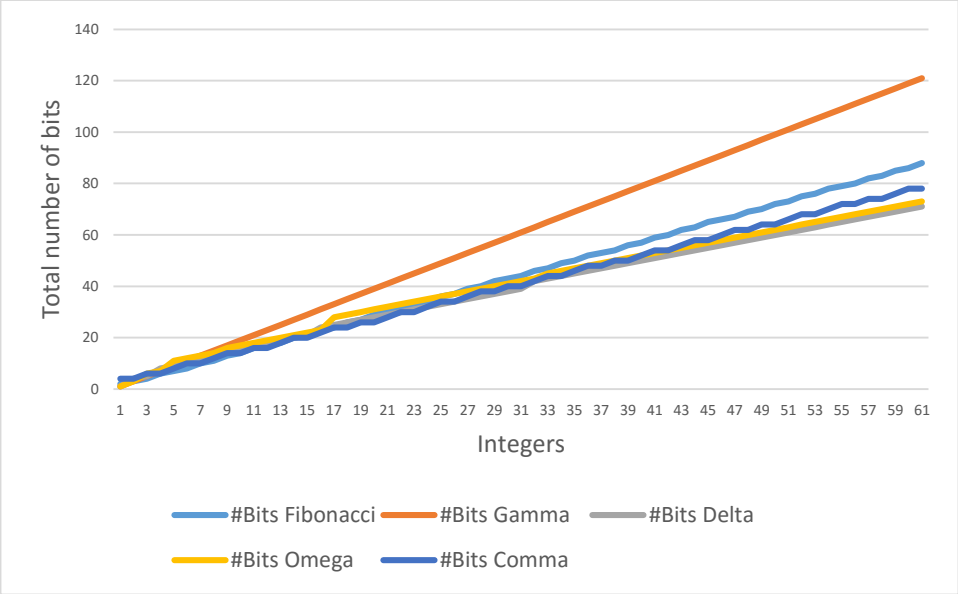


Figure 1.2(a): The total number of bits of the compression algorithms – $2^1 to\ 2^{60}$

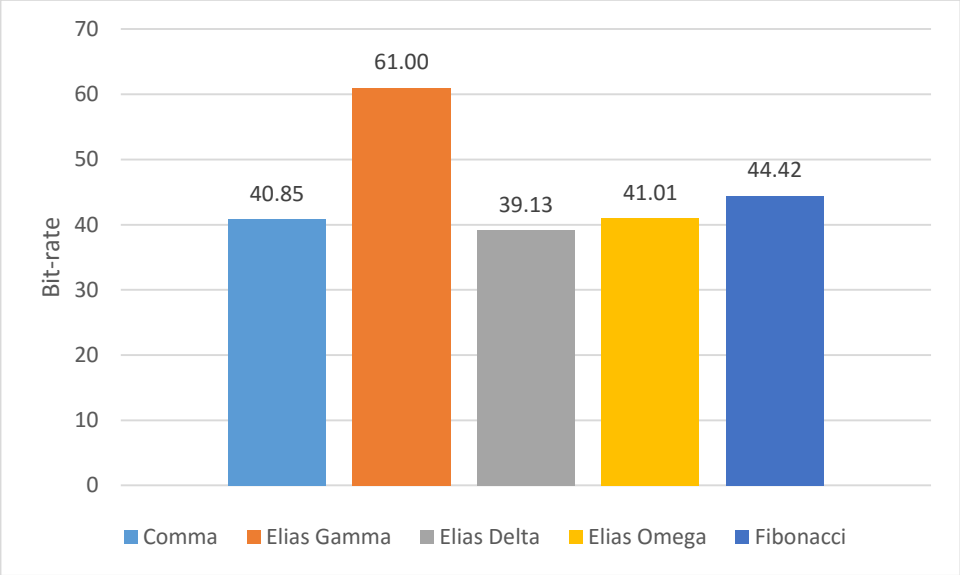


Figure 1.2(b): The bit-rate of the compression algorithms – $2^1 to 2^{60}$

From figures 1.2(a) and 1.2(b) we can observe that the Elias Delta code is the best compressor in this experiment with an average of 39.13 bits per integer. Comma code is the second best compressor with an average of 40.85 bits per integer. Whereas, Elias Gamma code is the lowest performer in this experiment with an average of 61.00 bits per integer.

4.3.2 Experiment 2

In this experiment, we have used the data-sets of GPDF with the probabilities of 0.5, 0.1, 0.01, and PD with $\lambda = 128$.

In previous experiments, we did not implement δ-Huffman code, because the data-sets do not have any repetition of input integers. Hence δ-Huffman code provides a result that is inferior to fixed length coding. From the current experiment and on we include the δ-Huffman code in the reported results. For all the data-sets used in this section of experiments, we provide the histograms, the bit-rate of the Golomb coding and the bit-rate of the Comma code, Elias Gamma code, Elias Delta code, Elias Omega code, Fibonacci code, and δ-Huffman code. At the end of the experiment, we provide observations and analysis of the results.

Figure 2.1 (a) provides the histogram of GPDF (0.5). Figures 2.1(b) – 2.1(c) provide the bit-rate of the Golomb coding and compression algorithms such as Comma code, Elias Gamma code, Elias Delta code, Elias Omega code, Fibonacci code, and δ-Huffman code.

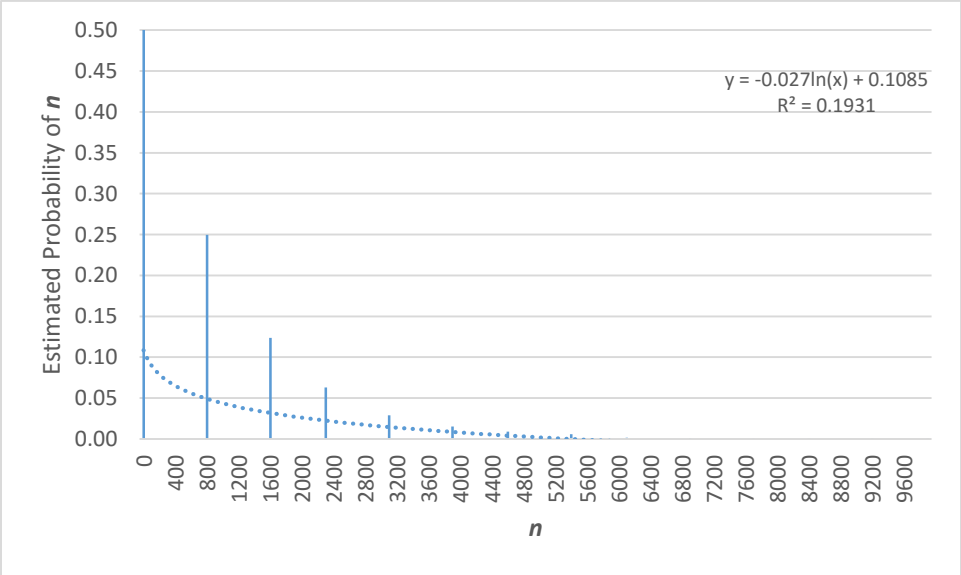


Figure 2.1(a): The histogram of GPDF (0.5)

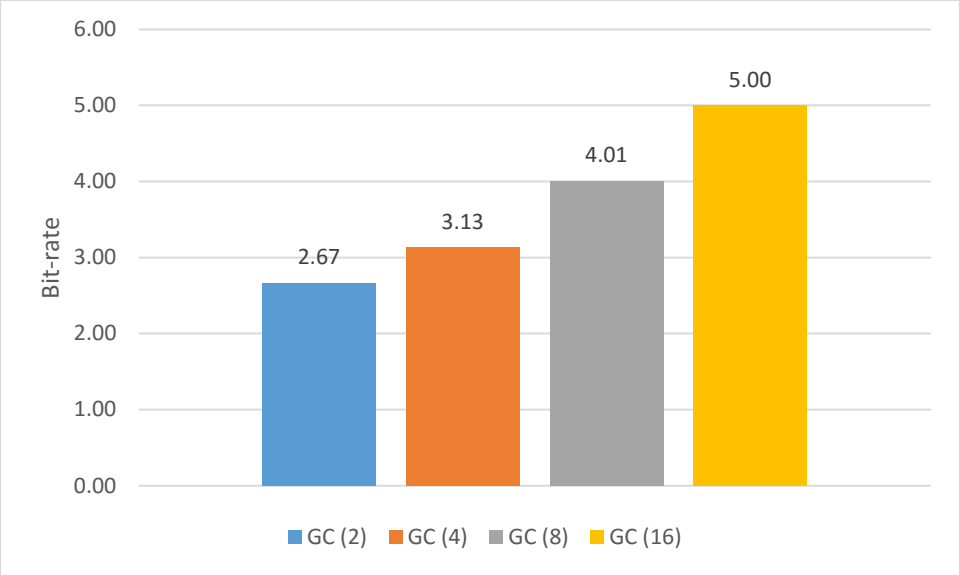


Figure 2.1(b): The bit-rate of the Golomb coding - GPDF (0.5)

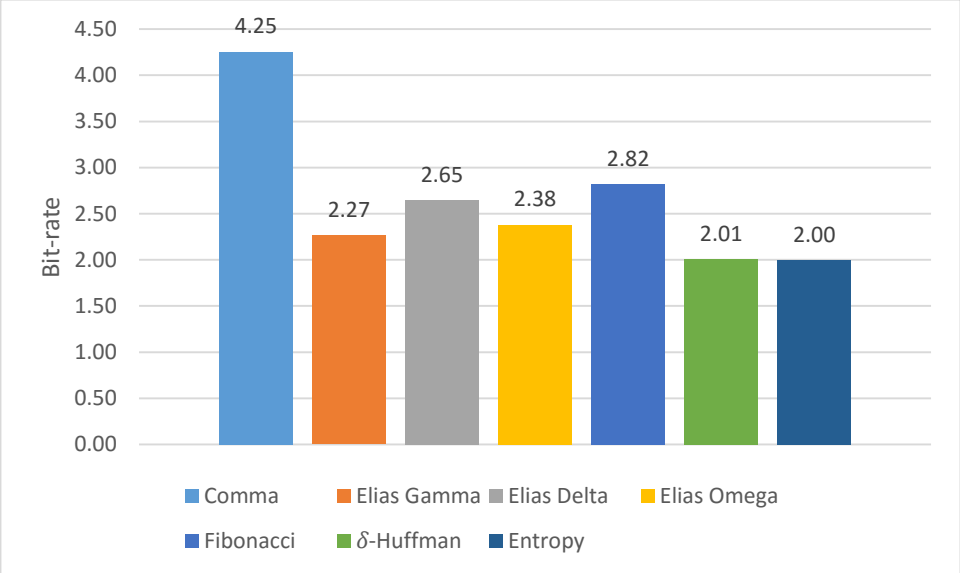


Figure 2.1(c): The bit-rate of the compression algorithms- GPDF (0.5)

From figure 2.1(a) we can observe that the data is distributed in the first few bins and clustered around 0 and 1 bins. The distribution of the data decreases in the latter half of the bins. The data-set contains very small integers with significant amount of repetitions. The maximum of the input is 14 and the minimum is 1. The average value of the input data-set is 2.002, which is due to the fact that the data-set contains very small integers with numerous

repetitions of input integers. The equation of the curve is $y = -0.027ln(x) + 0.1085$. The R-squared value (see section 4.1) is $R^2 = 0.1931$. In general, the high the R-squared, it can be considered as a better fit with values of 0.5 considered as the minimum accepted and 0.7 and above as good fit. In this case, we can observe that the R^2 value is low, hence it is not a good fit.

Figure 2.2 (a) provides the histogram of GPDF (0.1). Figures 2.2(b) – 2.2(c) provide the bit-rate of the Golomb coding and compression algorithms such as Comma code, Elias Gamma code, Elias Delta code, Elias Omega code, Fibonacci code, and δ-Huffman code.

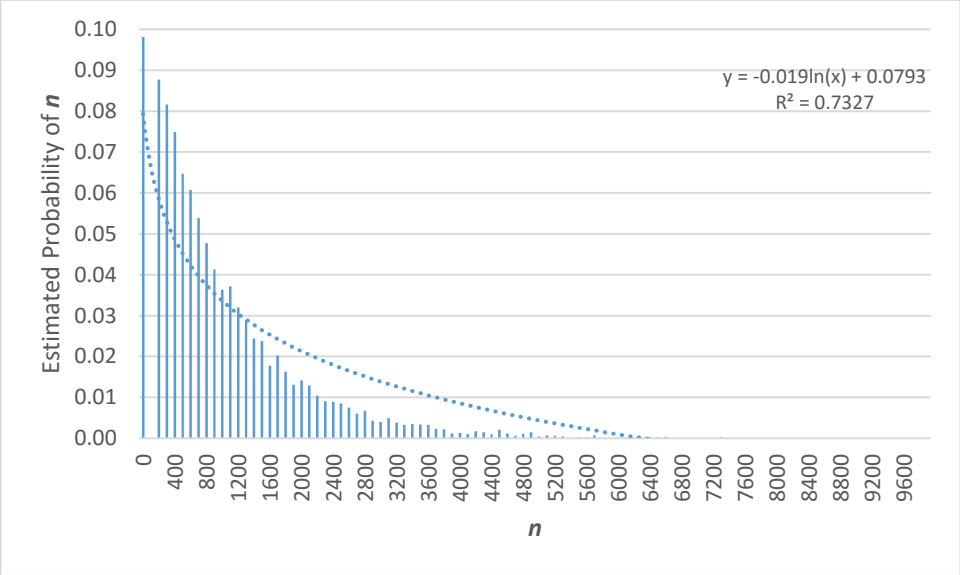


Figure 2.2(a): The histogram of GPDF (0.1)

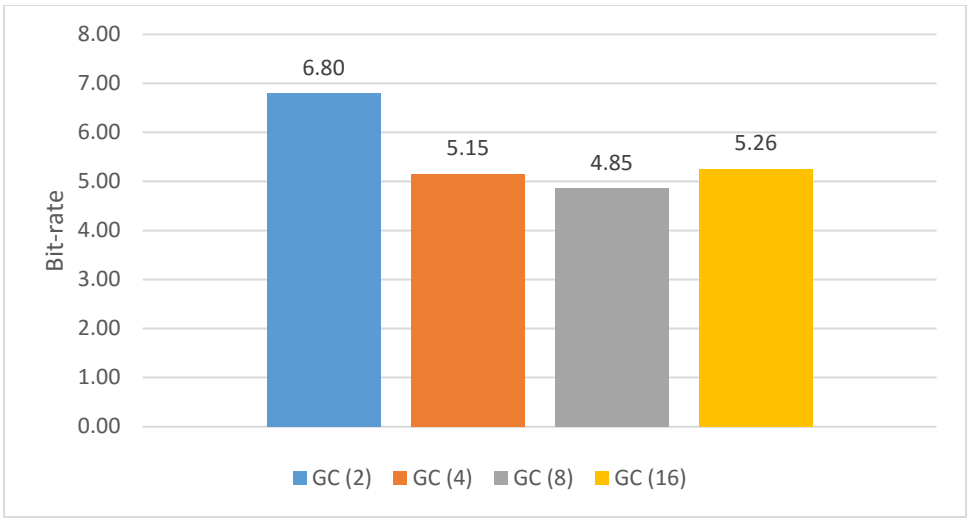


Figure 2.2(b): The bit-rate of the Golomb coding - GPDF (0.1)

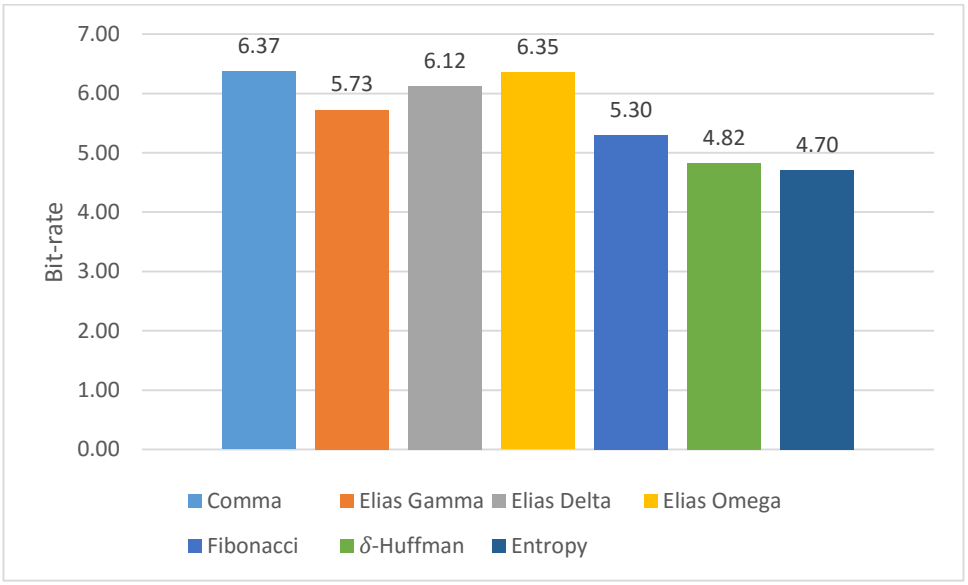


Figure 2.2(c): The bit-rate of the compression algorithms - GPDF (0.1)

From figure 2.2(a) we can observe that the data is distributed in the initial bins and clustered near the first 10 bins. The data-set contains medium small integers with repetitions. The maximum of the input is 99 and minimum is 1. The averages of the input data-set are 10.13, from which it can be observed that the data-set contains medium small integers with numerous repetitions. The equation of the curve is $y = -0.019ln(x) + 0.0793$. The R^2 of the curve is 0.7327. In this case, the R^2 is high, so it can be considered as a good fit. In the

latter experiments, we have compared the equation of the curves of GPDF to the equations of the sorted inverted lists and used it to determine which of the equations of GPDF are close to the equations of gaps of sorted inverted lists.

Figure 2.3 (a) provides the histogram of GPDF (0.01). Figures 2.3(b) – 2.3(c) provide the bit-rate of the Golomb coding and compression algorithms such as Comma code, Elias Gamma code, Elias Delta code, Elias Omega code, Fibonacci code, and δ-Huffman code.

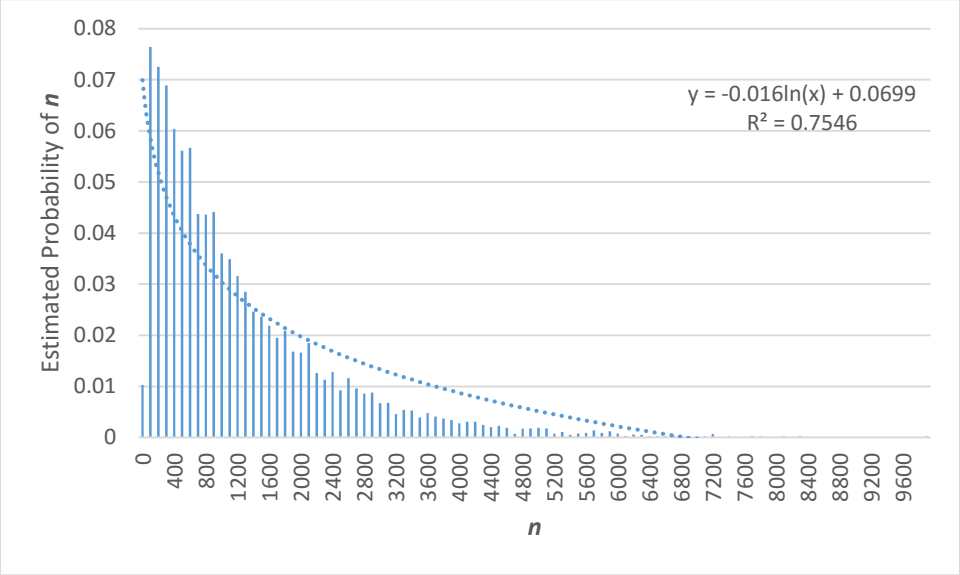


Figure 2.3(a): The histogram of GPDF (0.01)

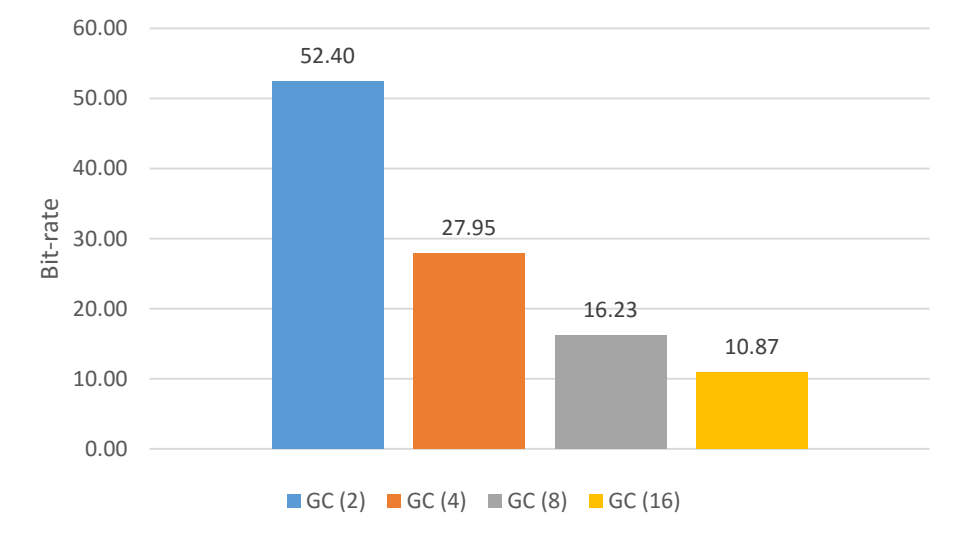


Figure 2.3(b): The bit-rate of the Golomb coding - GPDF (0.01)

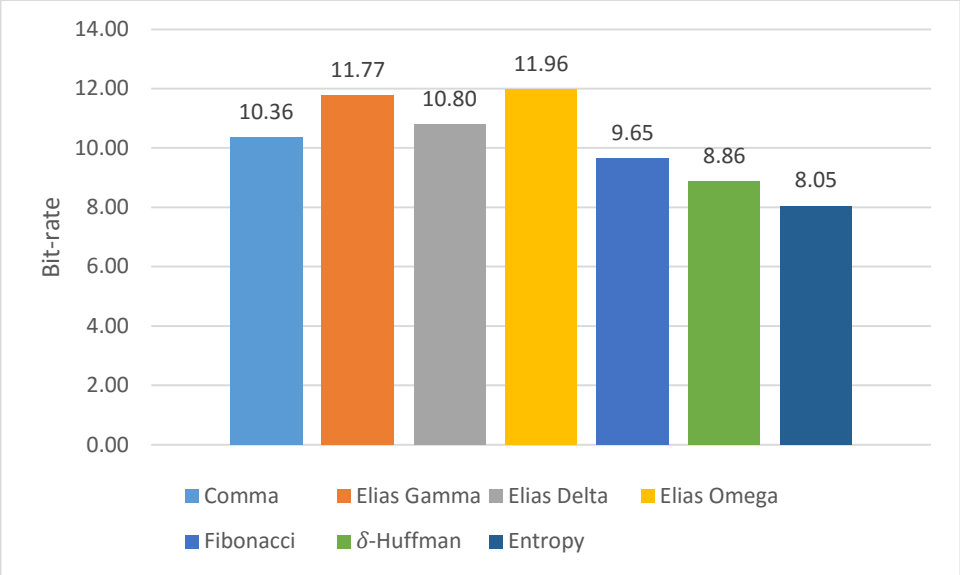


Figure 2.3(c): The bit-rate of the compression algorithms – GPDF (0.01)

From figure 2.3(a) we can observe that the data is distributed in the bins near 0 and 100. The maximum of the input is 826 and minimum is 1. The averages of the input data-set are 101.29, from which it can be observed that the data-set contains medium small to medium large integers with numerous repetitions. The equation of the curve is $y = -0.016ln(x) + 0.0699$. The R^2 of the curve is 0.7546. Among all the GPDF with different probabilities, Geometric distribution with probability of 0.01 has the high R^2 value and it is the best fit.

Figure 2.4 (a) provides the histogram of PD. Figures 2.4(b) – 2.4(c) provide the bit-rate of the Golomb coding and compression algorithms such as Comma code, Elias Gamma code, Elias Delta code, Elias Omega code, Fibonacci code, and δ-Huffman code.

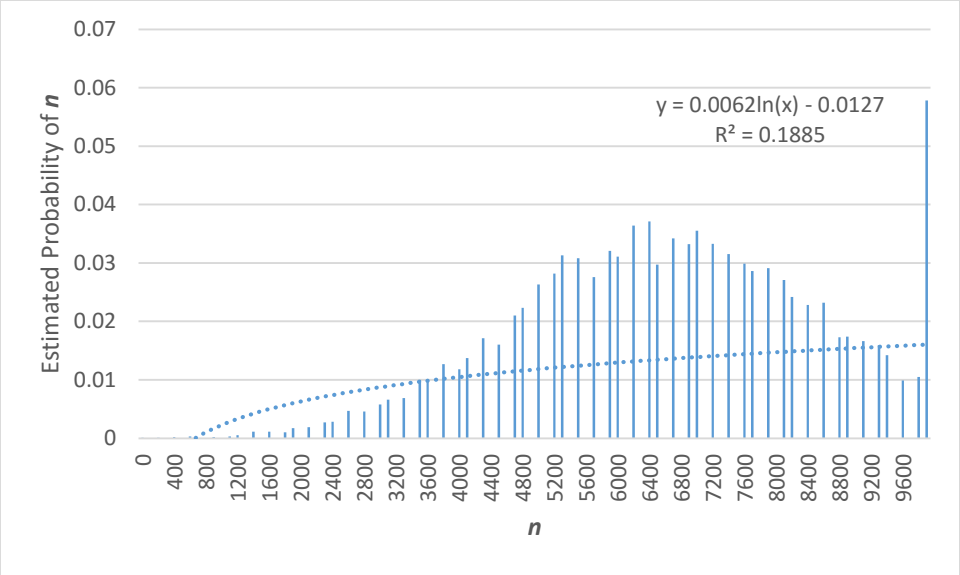


Figure 2.4(a): The histogram of Poisson Distribution (PD)

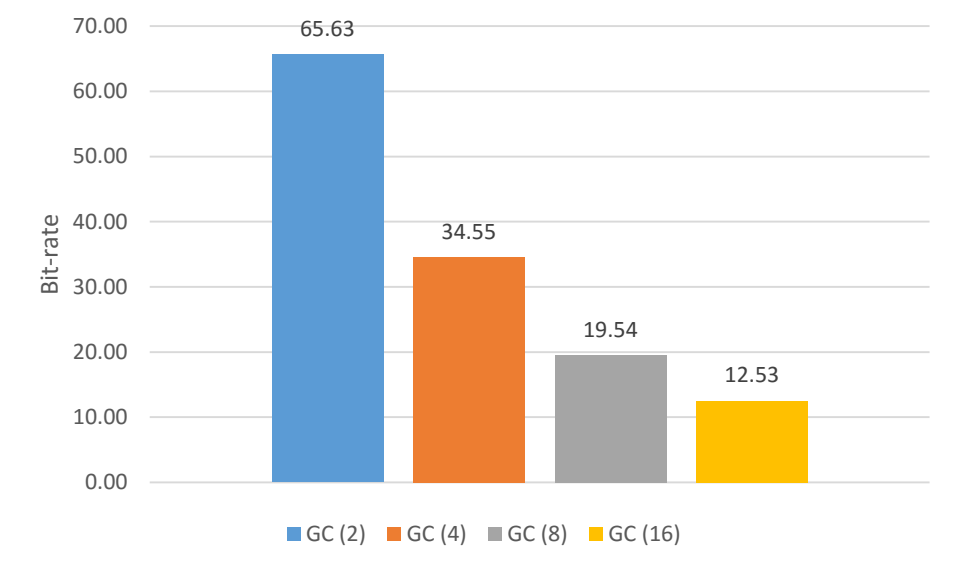


Figure 2.4(b): The bit-rate of the Golomb coding - PD

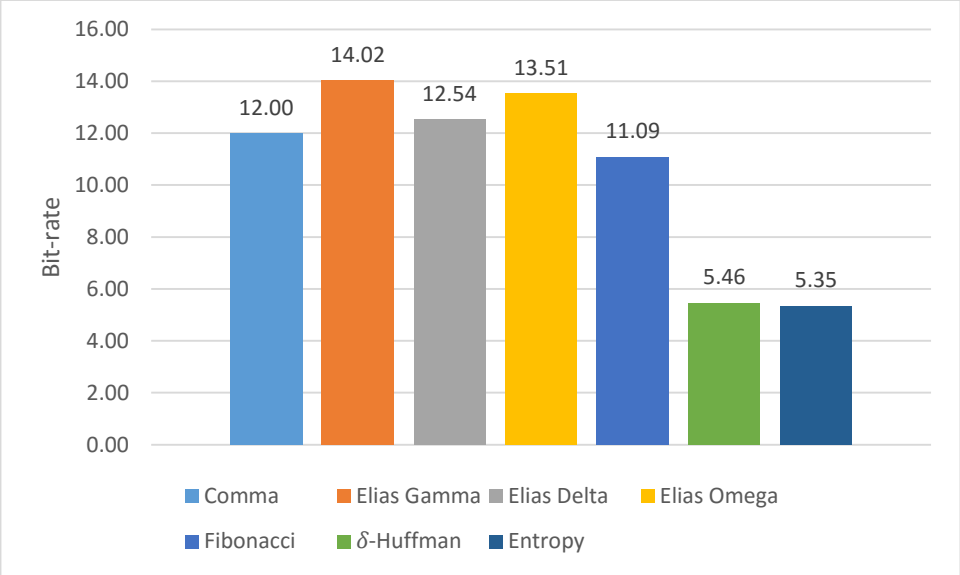


Figure 2.4(c): The bit-rate of the compression algorithms - PD

From figure 2.4(a) we can observe that the distribution of the data is not concentrated in the initial bins. The data is distributed from the middle segment of bins near 112 and 140. The maximum of the input is 147 and minimum is 89. The averages of the input data-set are 127.78, from which it can be observed that the data-set contains medium to medium large integers with numerous repetitions. The equation of the curve is $y = 61.651ln(x) - 125.84$. The R^2 of the curve is 0.1853. With the low R^2 value of Poisson, it is not a good fit.

From the results of GPDF with the probabilities of 0.5, 0.1, and 0.01 we can observe that the data-set of GPDF (0.5) contains very small integers with numerous repetitions. Whereas, the other data-set of GPDF (0.1) contains medium size integers with less repetitions and GPDF with the probability of 0.01 contains larger input integers with few repetitions. In all of the probabilities, regardless of the integer distribution there are quite a lot of repetitions, δ-Huffman code provides the low bit-rate in data-sets when numerous repetitions of small integers are involved. This is because, whenever the repetitions occur the δ-Huffman sends the value of the repeated integers from the Huffman tree which are already updated on the

tree and requires small number of bits to encode compared to the encoding of new integers. In all of these cases, it can be observed that δ-Huffman code provides the best bit-rate which is relatively close to the entropy.

Compared to the PD and GPDF, the PD has integers with the highest average value. In spite of this, its entropy is relatively low and the δ-Huffman code requires an average of 5.46 bits per integer which is close to the entropy of 5.35 bits per integer. Among all the distributions of Geometric and Poisson, δ-Huffman code provides the highest bit-rate in GPDF (0.01). Whereas, for GPDF (0.5) δ-Huffman code provides the lowest bit-rate.

We have drawn the data-sets from GPDF and PD, since they have been described in the literature as good approximations for realistic data, in specific GPDFs provide good approximation for the gaps of inverted lists. In all the cases δ-Huffman code provides the best bit-rate which is relatively close to the entropy compared to the other compression algorithms.

4.3.3 Experiment 3

In this experiment, we have used the data-sets of gaps of GPDF with the probabilities of 0.5, 0.1, 0.01, and gaps of PD with $\lambda = 128$. For the differences, we have applied the sign and magnitude representation and the odd-even mapping (except for GPDF (0.5)). In all of the following figures in this section, [S-M] refers to the sign and magnitude representation and [O-E] refers to the odd-even mapping.

For all the data-sets used in this section of the experiments, we have provided the bit-rate of the Golomb coding and the bit-rate of the Comma code, Elias Gamma code, Elias Delta code, Elias Omega code, Fibonacci code, and δ-Huffman code. At the end of the experiment, we have provided observations and analysis of the results.

Figures 3.1(a) – 3.1(b) provide the bit-rate of the Golomb coding and compression algorithms such as Comma code, Elias Gamma code, Elias Delta code, Elias Omega code,

Fibonacci code, and δ-Huffman code.

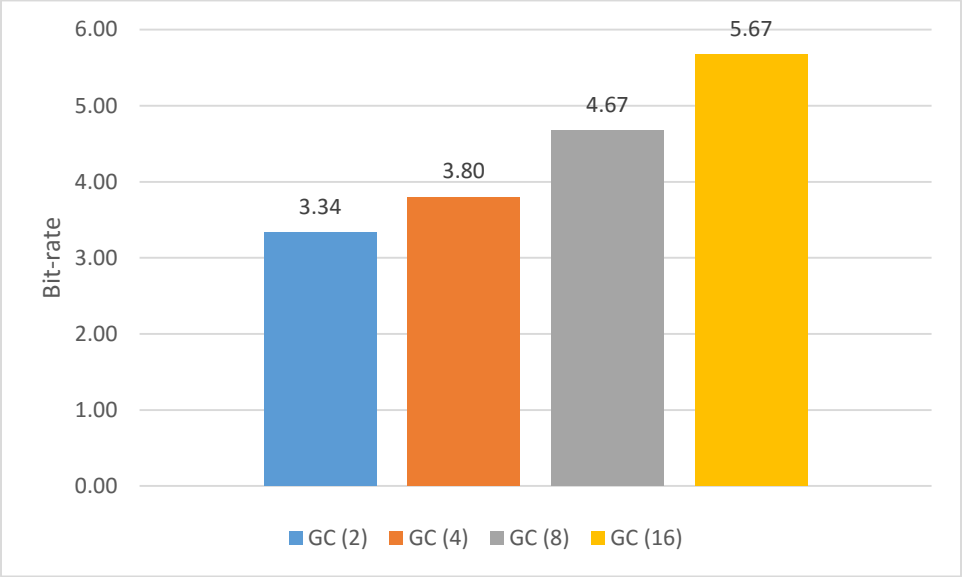


Figure 3.1(a): The bit-rate of the Golomb coding – Gaps of GPDF (0.5) [S-M]

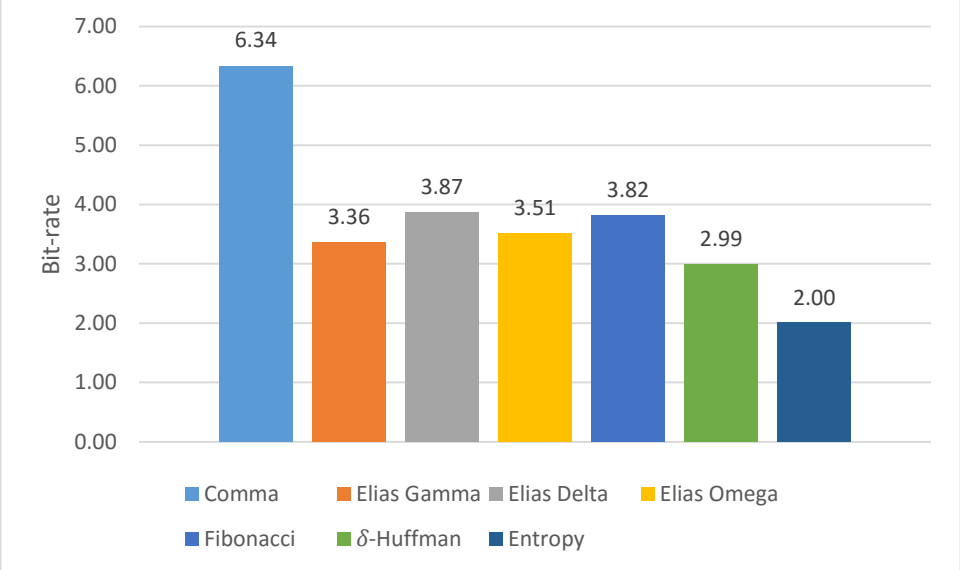


Figure 3.1(b): The bit-rate of the compression algorithms – Gaps of GPDF (0.5) [S-M]

From the results of GPDF (0.5) to the gaps of GPDF (0.5), we can observe that the δ-Huffman code requires fewer bits to compress GPDF (0.5) than the gaps of GPDF (0.5). In the gaps, the integers are in closer intervals with numerous repetitions. In both of the cases, δ-Huffman code provides the best bit-rate which is relatively close to the entropy.

Figures 3.2(a) – 3.2(b) provide the bit-rate of the Golomb coding and compression algorithms such as Comma code, Elias Gamma code, Elias Delta code, Elias Omega code, Fibonacci code, and δ-Huffman code.

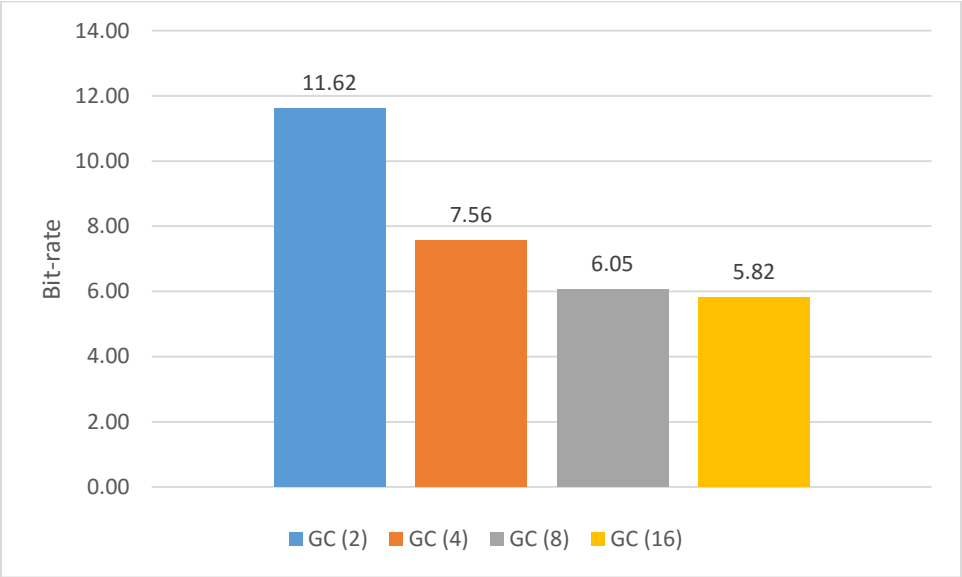


Figure 3.2(a): The bit-rate of the Golomb coding – Gaps of GPDF (0.1) [O-E]

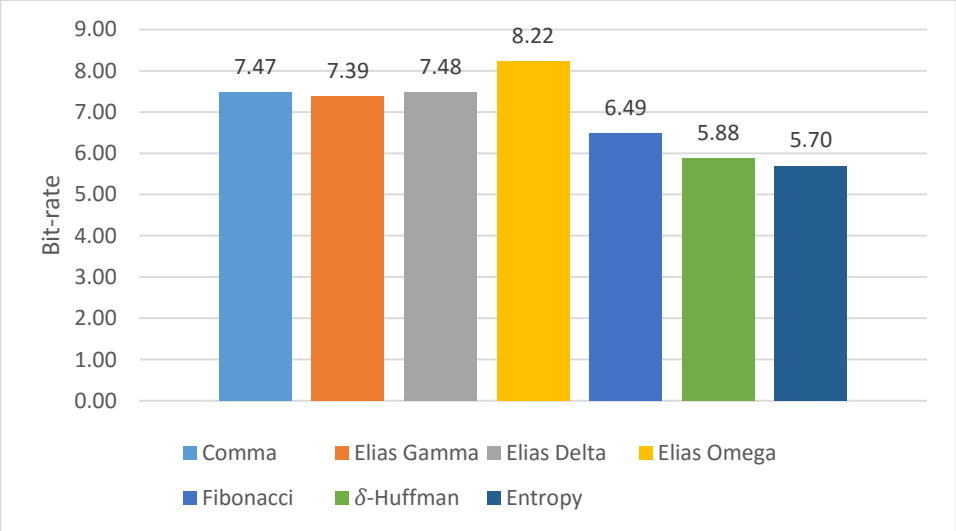


Figure 3.2(b): The bit-rate of the compression algorithms – Gaps of GPDF (0.1) [O-E]

From the results of GPDF (0.1) to the gaps of GPDF (0.1), we can observe that the δ-Huffman code requires less bits for compressing GPDF (0.1) than the gaps of GPDF (0.1).

Yet in both of the cases δ-Huffman code provides the best bit-rate which is relatively close to the entropy.

Figures 3.3(a) – 3.3(b) provide the bit-rate of the Golomb coding and compression algorithms such as Comma code, Elias Gamma code, Elias Delta code, Elias Omega code, Fibonacci code, and δ-Huffman code.

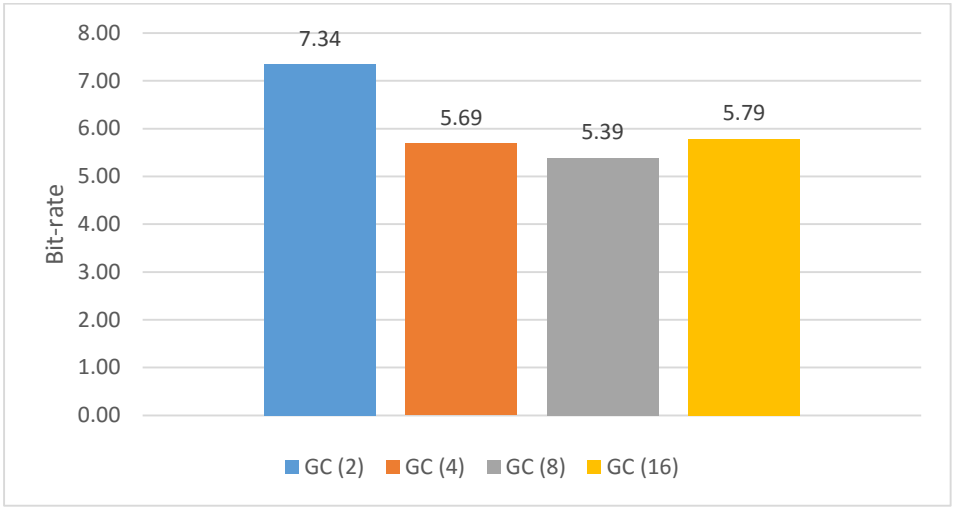


Figure 3.3(a): The bit-rate of the Golomb coding – Gaps of GPDF (0.1) [S-M]

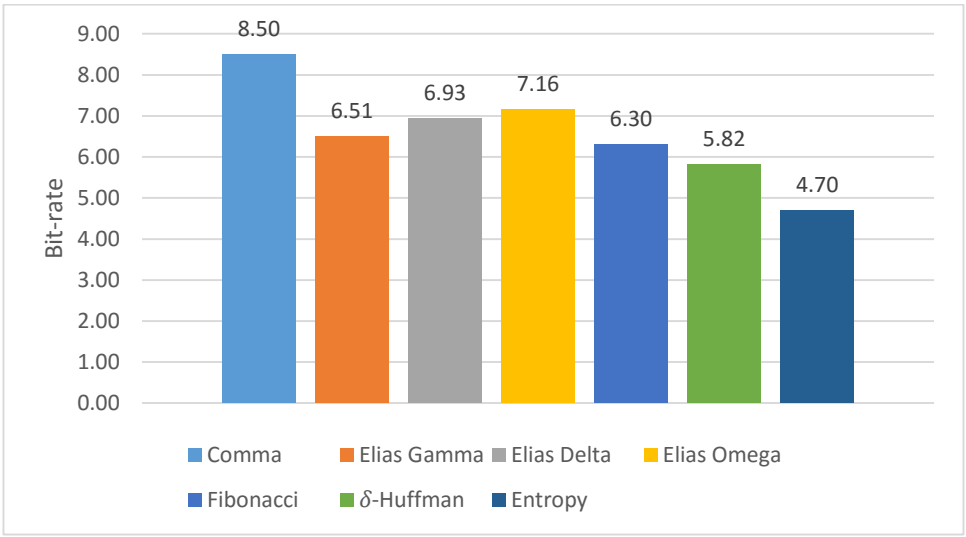


Figure 3.3(b): The bit-rate of the compression algorithms – Gaps of GPDF (0.1) [S-M]

From the experimental results reported in sections 3.2 and 3.3, we can observe that the data-sets are drawn from the gaps of GPDF (0.1). But in each experiment the gaps are

treated differently. We can observe that mapping the differences to the odd-even integers gives an entropy of 5.70 bits per integer, whereas using the sign and magnitude representation gives an entropy of 4.70 bits per integer. From this, we can conclude that the sign and magnitude representation is better for the compression of negative differences than implementing the odd-even mapping.

Figures 3.4(a) – 3.4(b) provide the bit-rate of the Golomb coding and compression algorithms such as Comma code, Elias Gamma code, Elias Delta code, Elias Omega code, Fibonacci code, and δ-Huffman code.

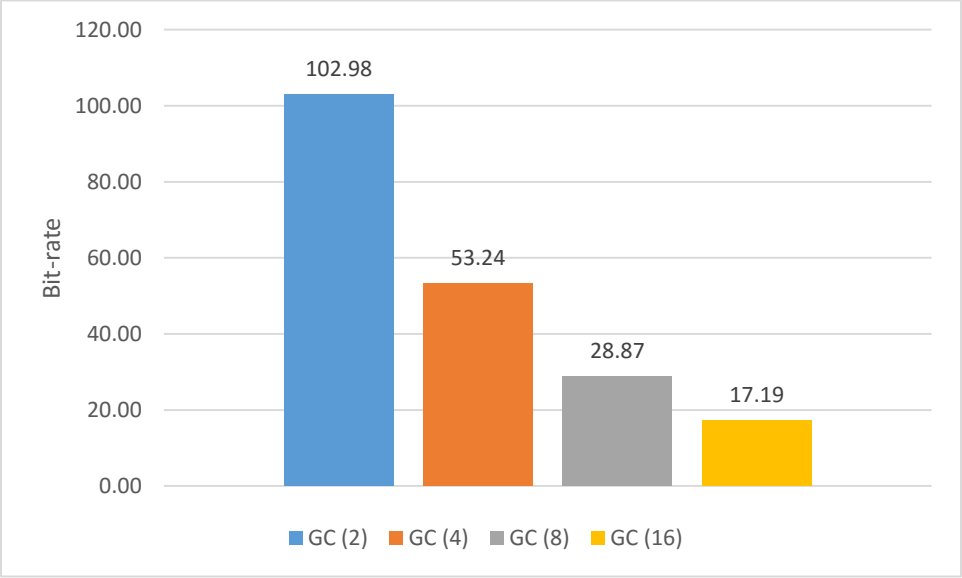


Figure 3.4(a): The bit-rate of the Golomb coding – Gaps of GPDF (0.01) [O-E]

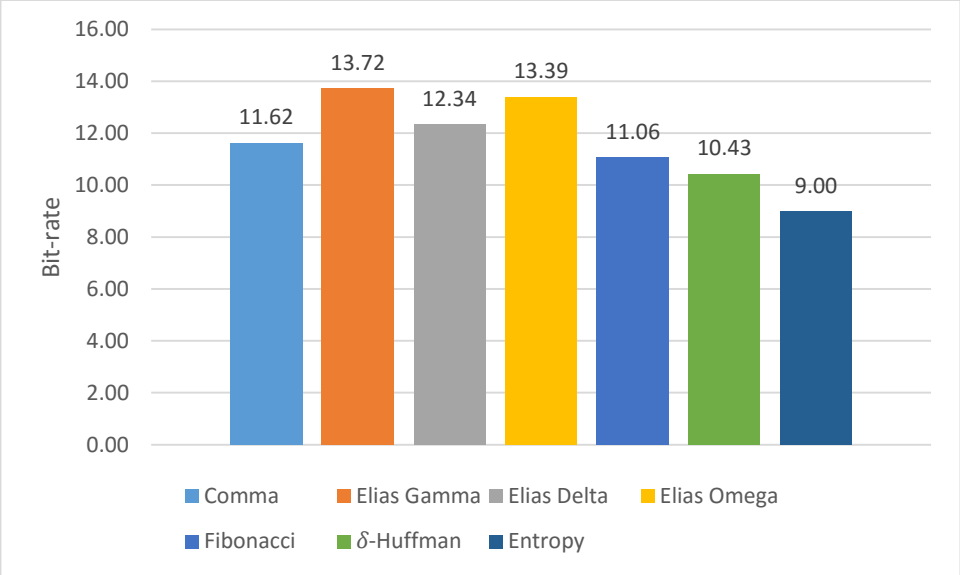


Figure 3.4(b): The bit-rate of the compression algorithms – Gaps of GPDF (0.01) [O-E]

Figures 3.5(a) – 3.5(b) provide the bit-rate of the Golomb coding and compression algorithms such as Comma code, Elias Gamma code, Elias Delta code, Elias Omega code, Fibonacci code, and δ-Huffman code.

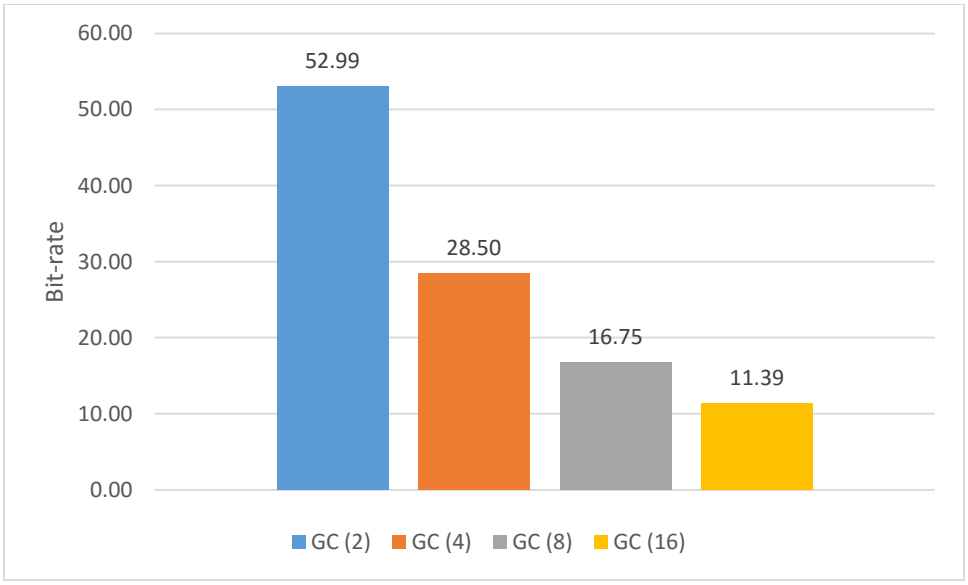


Figure 3.5(a): The bit-rate of the Golomb coding – Gaps of GPDF (0.01) [S-M]

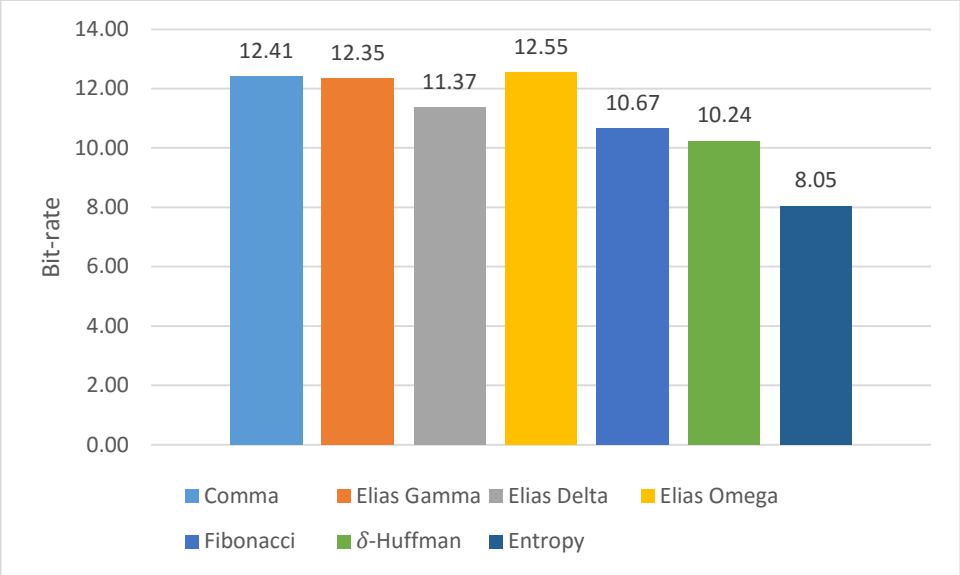


Figure 3.5(b): The bit-rate of the compression algorithms – Gaps of GPDF (0.01) [S-M]

From the experimental results reported in sections 3.4 and 3.5, we can observe that the data-sets are drawn from the gaps of GPDF (0.01). But in each experiment the gaps are treated differently. We can observe that by mapping the differences to the odd-even integers provides an entropy of 9.00 bits per integer, whereas using the sign and magnitude representation provides an entropy of 8.05 bits per integer. This explains that sign and magnitude representation provides the best compression rate for all the compression algorithms.

Figures 3.6(a) – 3.6(b) provide the bit-rate of the Golomb coding and compression algorithms such as Comma code, Elias Gamma code, Elias Delta code, Elias Omega code, Fibonacci code, and δ-Huffman code.

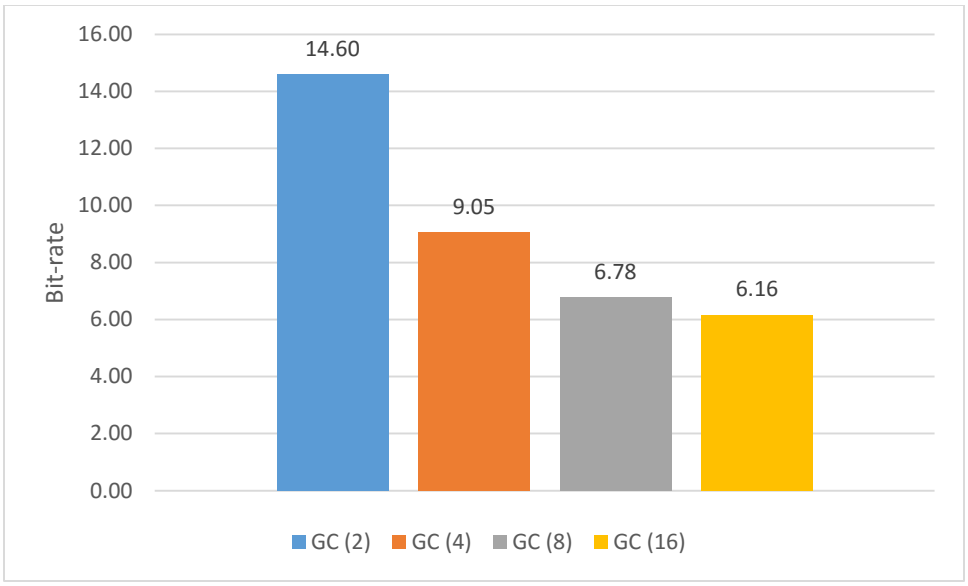


Figure 3.6(a): The bit-rate of the Golomb coding – Gaps of PD [O-E]

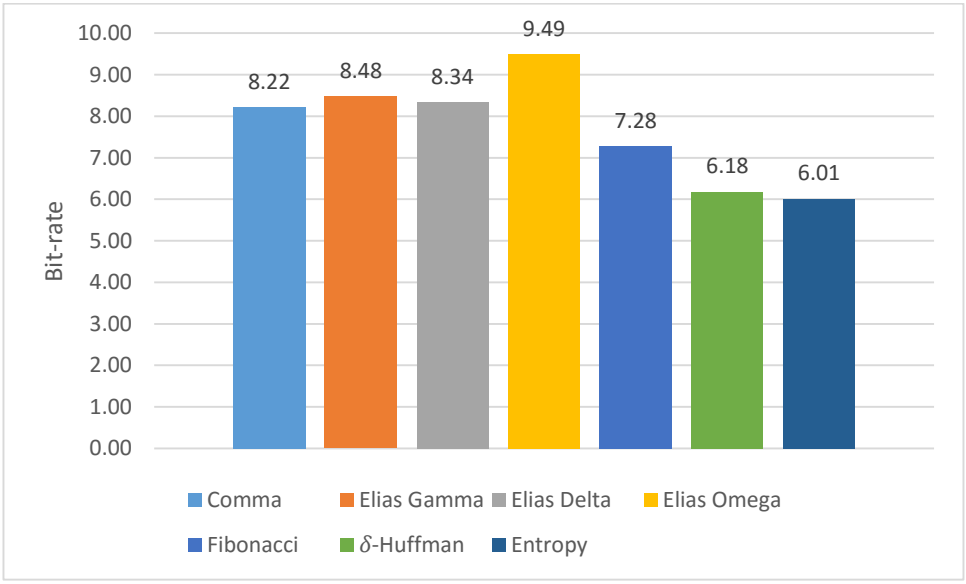


Figure 3.6(b): The bit-rate of the compression algorithms – Gaps of PD [O-E]

Figures 3.7(a) – 3.7(b) provide the bit-rate of the Golomb coding and compression algorithms such as Comma code, Elias Gamma code, Elias Delta code, Elias Omega code, Fibonacci code, and δ-Huffman code.

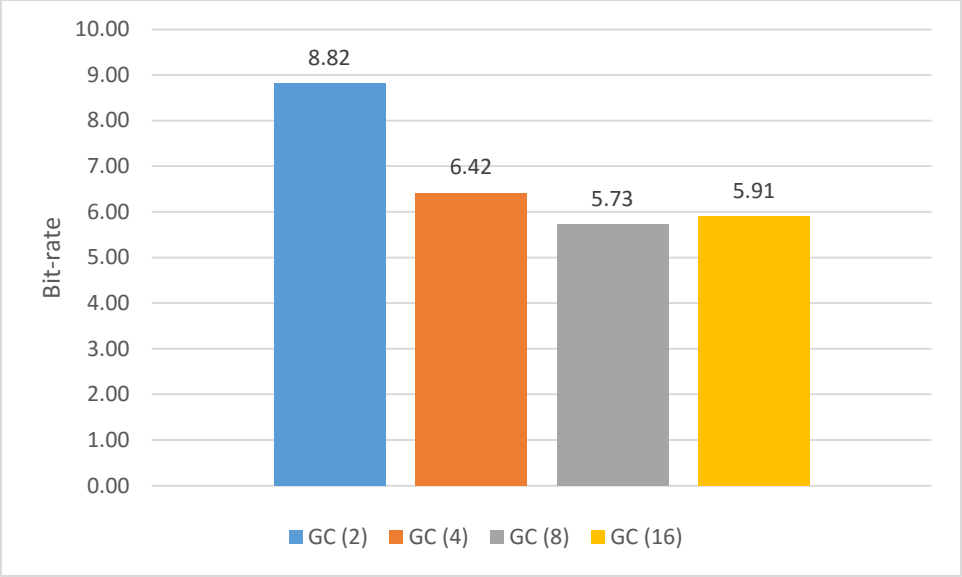


Figure 3.7(a): The bit-rate of the Golomb coding – Gaps of PD [S-M]

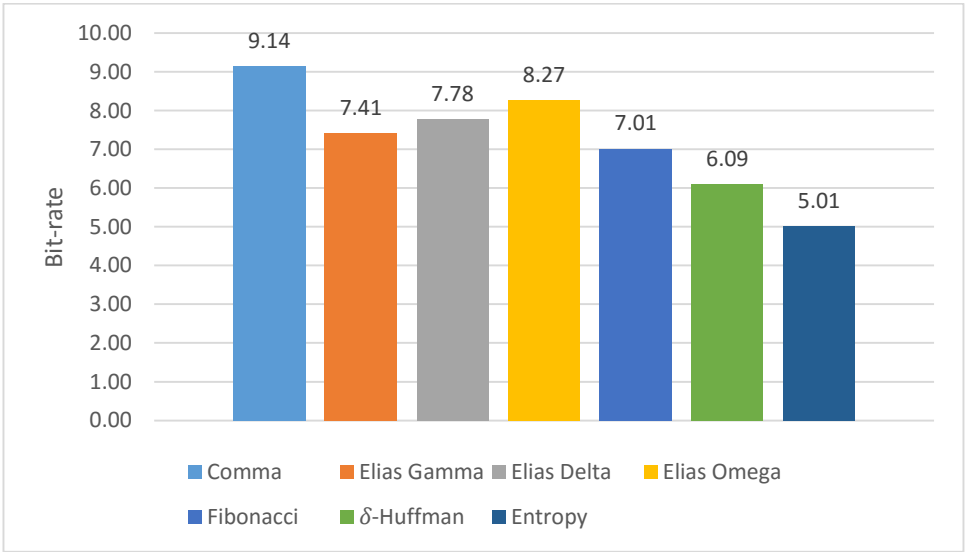


Figure 3.7(b): The bit-rate of the compression algorithms – Gaps of PD [S-M]

From the experimental results reported in sections 3.6 and 3.7, we can observe that the data-sets are drawn from the gaps of Poisson Distribution with $\lambda = 128$. But in each experiment the gaps are treated differently. We can observe that mapping the differences to odd-even integers provides an entropy of 6.01 bits per integer, whereas using the sign and magnitude representation provides an entropy of 5.01 bits per integer.

4.3.4 Experiment 4

In this experiment, we have used the data-sets of gaps of gaps of GPDF with probabilities of 0.5, 0.1, 0.01, and PD with $\lambda = 128$. Along with, we have compared compression techniques Elias Delta code, δ-Huffman code, and the entropy on the gaps vs. gaps of gaps. For the differences, we have used the sign and magnitude representation (see section 4.2.1). This is because, the sign and magnitude representation provides better compression than the odd-even mapping. In all of the following graphs, "O" refers to the data-set of the original distributions, "G" refers to gaps of distributions and "GG" refers to gaps of gaps of distributions.

Figures 4(a), 4(b), 4(c), and 4(d) provide the bit-rate of the Elias Delta code, δ-Huffman code, and entropy performed on gaps of gaps of GPDF with probabilities of 0.5, 0.1, 0.01 and PD with $\lambda = 128$.

In all of the following graphs in this experiment, the left column bar (light blue) refers to the compression techniques performed on the original data-set, the middle column bar (Gray) to the compression techniques performed on the gaps of the data-set and the right column bar (dark blue) refers to the compression techniques performed on the gaps of gaps of the data-set.

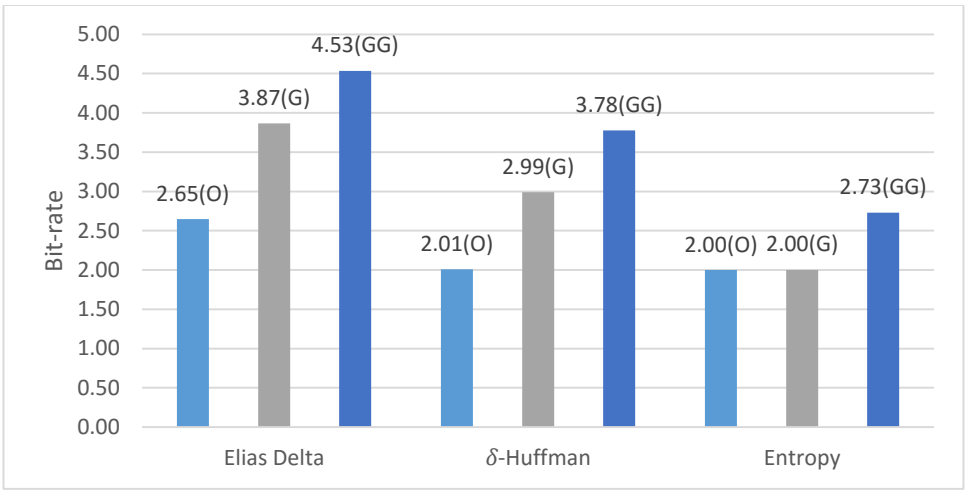


Figure 4(a): The bit-rate of GPDF (0.5) original vs. gaps vs. gaps of gaps

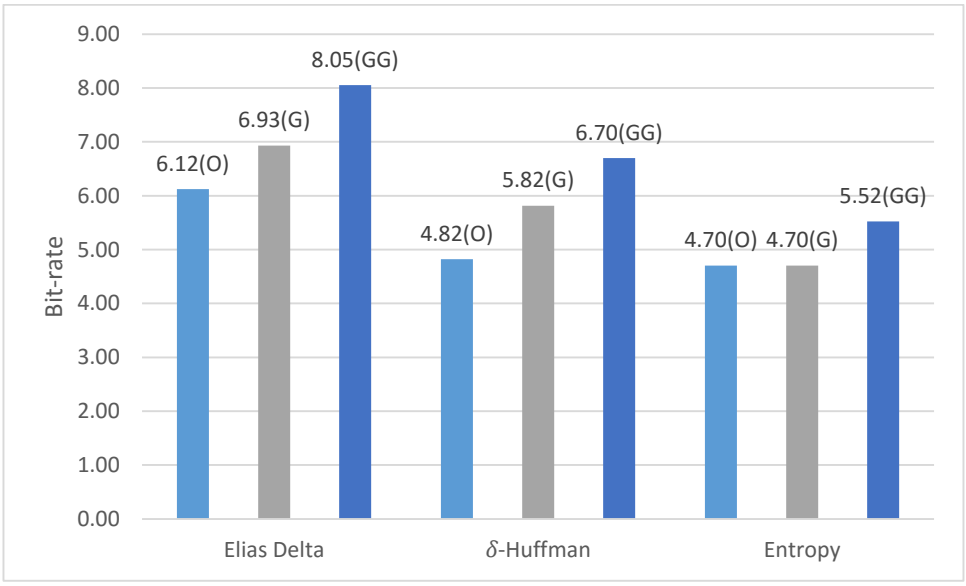


Figure 4(b): The bit-rate of GPDF (0.1) original vs. gaps vs. gaps of gaps

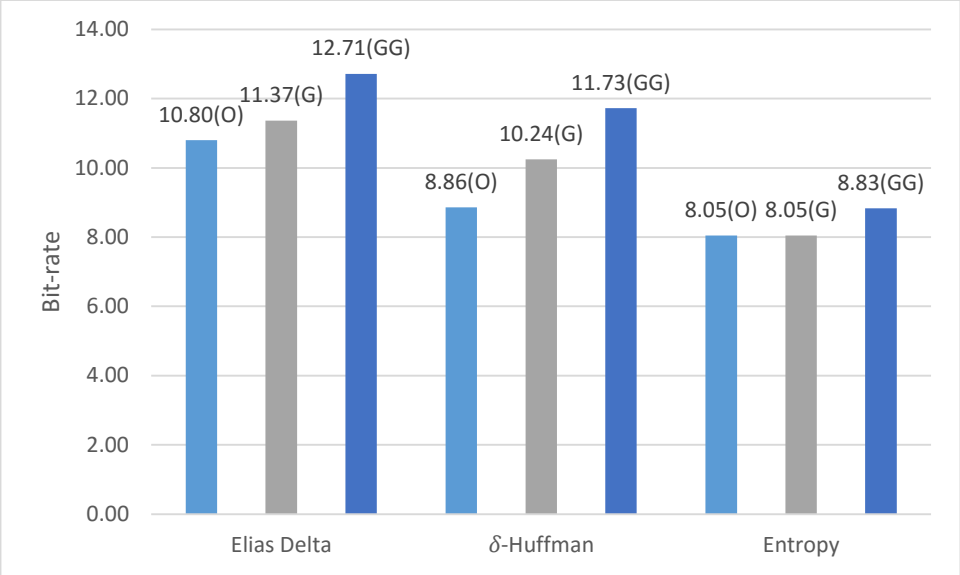


Figure 4(c): The bit-rate of GPDF (0.01) original vs. gaps vs. gaps of gaps

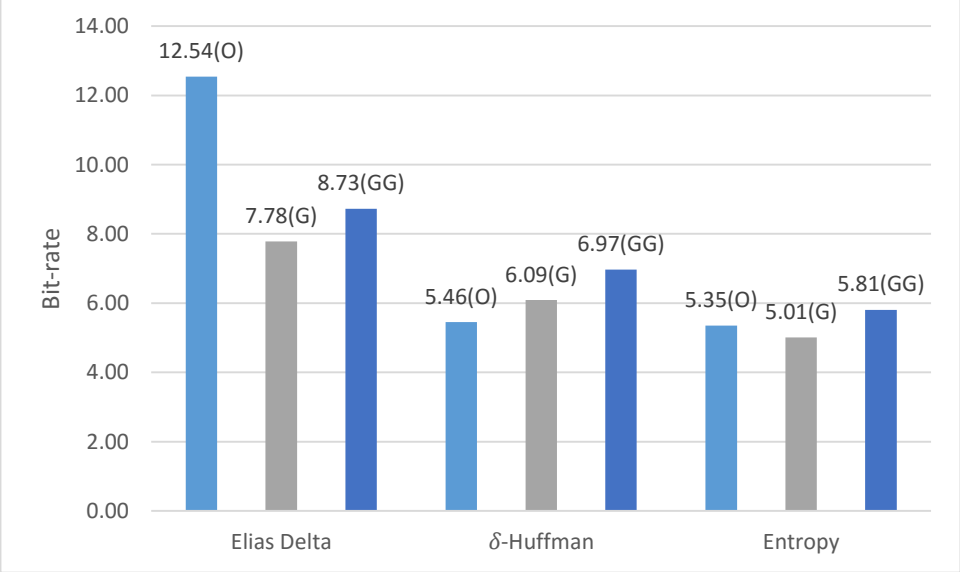


Figure 4(d): The bit-rate of PD original vs. gaps vs. gaps of gaps

From figures 4(a), 4(b), 4(c), and 4(d) we can observe that for all of the GPDF's, better bit-rate can be achieved by applying compression techniques on the original data-set rather than implementing compression techniques on the gaps or gaps of gaps. The reason is that the original data-set of GPDF contains integers in close intervals with numerous repetitions. By applying gaps or gaps of gaps on GPDF, the repetitions become smaller compared to the

original data-set and the gaps are widely separated. But in the case of PD, the original data-set contains large integers with less repetitions. By applying gaps, the integers are in closer intervals with numerous repetitions. This has provided the best bit-rate compared to the original data-set of PD or gaps of gaps of PD. In all the cases of GPDF and PD, δ-Huffman code provides the best bit-rate relatively close to the entropy.

4.3.5 Experiment 5

In this experiment, we have used the data-set of gaps of sorted inverted lists obtained from Wikipedia [from the year 2015].

For all the data used in this section of experiments, we provide the bit-rate of the Golomb coding and the bit-rate of the Comma code, Elias Gamma code, Elias Delta code, Elias Omega code, Fibonacci code, and δ-Huffman code. At the end of the experiment, we provide observations and analysis of the results.

Figure 5.1 (a) provides the histogram of "2015". Figures 5.1(b) – 5.1(c) provide the bit-rate of the Golomb coding and compression algorithms such as Comma code, Elias Gamma code, Elias Delta code, Elias Omega code, Fibonacci code, and δ-Huffman code.

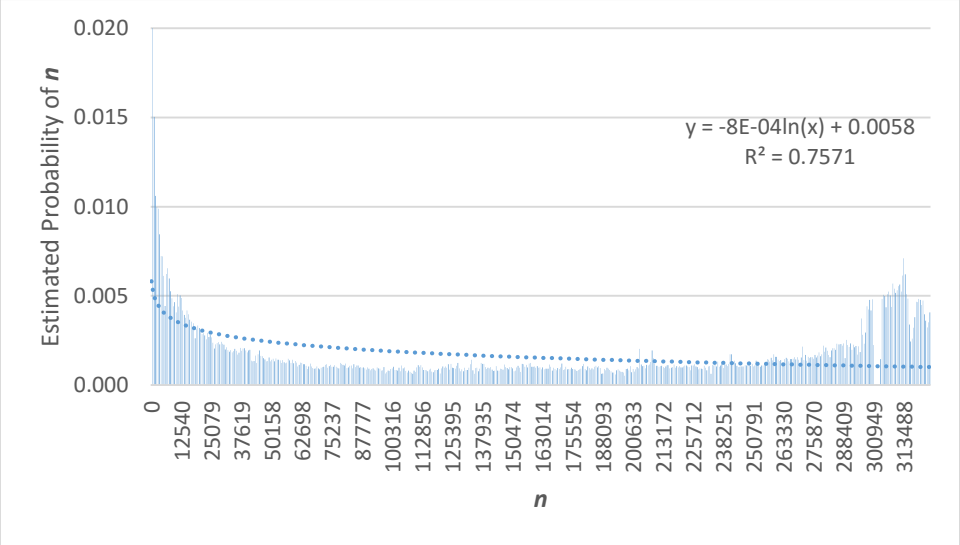


Figure 5.1(a): The histogram of 2015

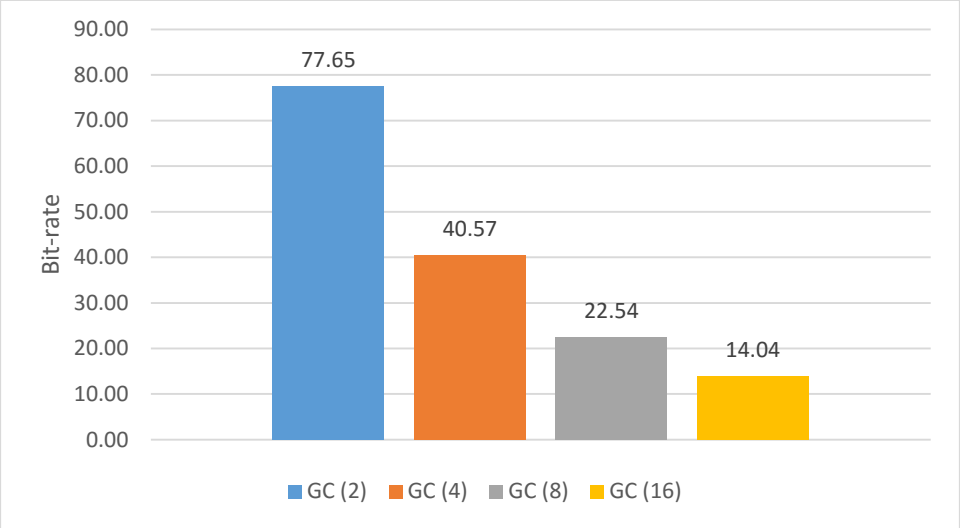


Figure 5.1(b): The bit-rate of the Golomb coding – 2015

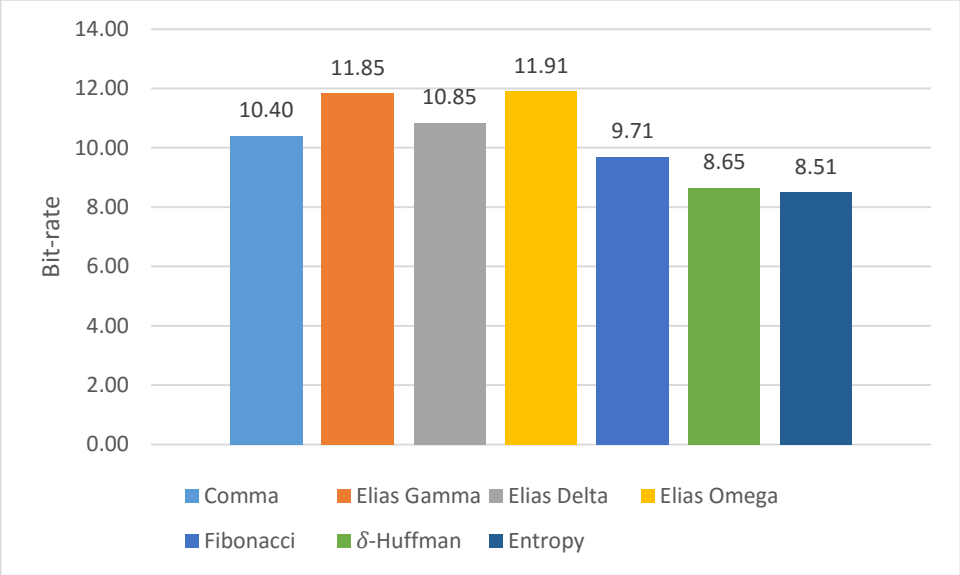


Figure 5.1(c): The bit-rate of the compression algorithms – 2015

The R-squared value of the "**2015**" histogram is 0.7571. Hence, it is a good fit. The equation of "**2015**" histogram, which is $y = -8E-04ln(x) + 0.0058$ is close to the equation of GPDF (0.01) which is $y = -0.016ln(x) + 0.0699$.

From the above equations, proximity we can expect that the data-set would have a δ-Huffman code averages within a close margin or range. The calculated average of the δ-Huffman codeof "**2015**" is 8.65 bits per integer. The calculated average of the δ-Huffman

code of GPDF (0.01) is 8.86 bits per integer. In this case, we can observe that the averages are almost equivalent.

Figure 5.2 (a) provides the histogram of "Bollywood". Figures 5.2(b) – 5.2(c) provides the bit-rate of the Golomb coding and compression algorithms such as Comma code, Elias Gamma code, Elias Delta code, Elias Omega code, Fibonacci code and δ-Huffman code.

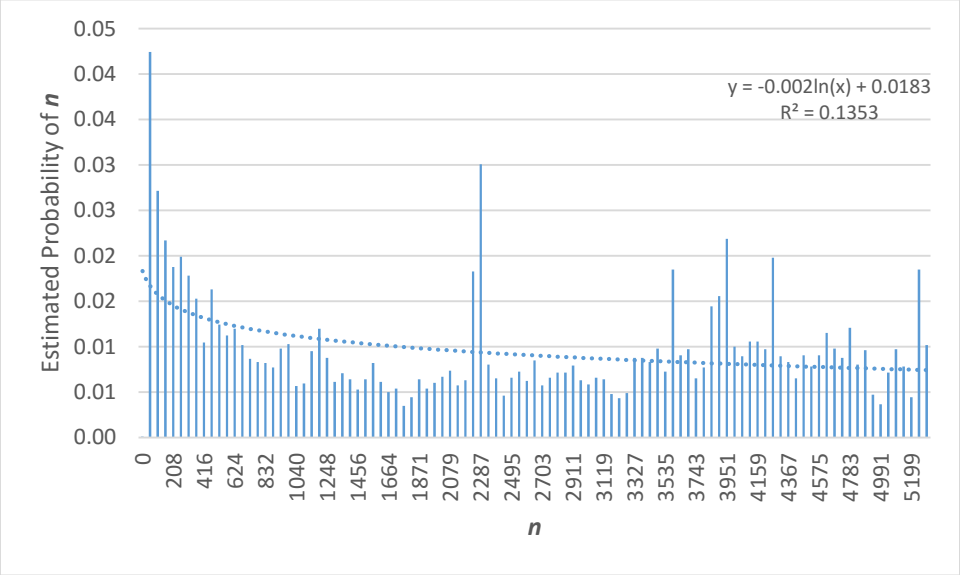


Figure 5.2(a): The histogram of Bollywood

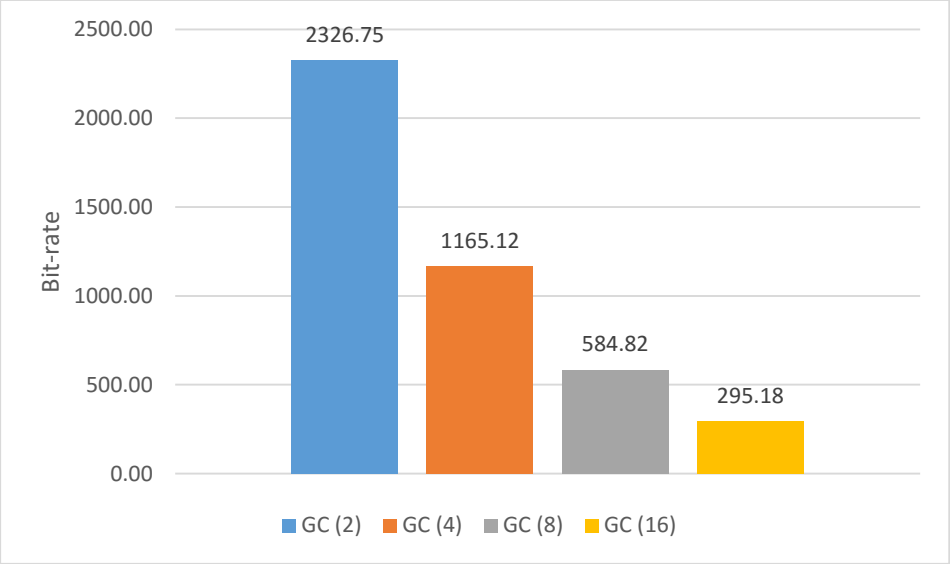


Figure 5.2(b): The bit-rate of the Golomb coding – Bollywood

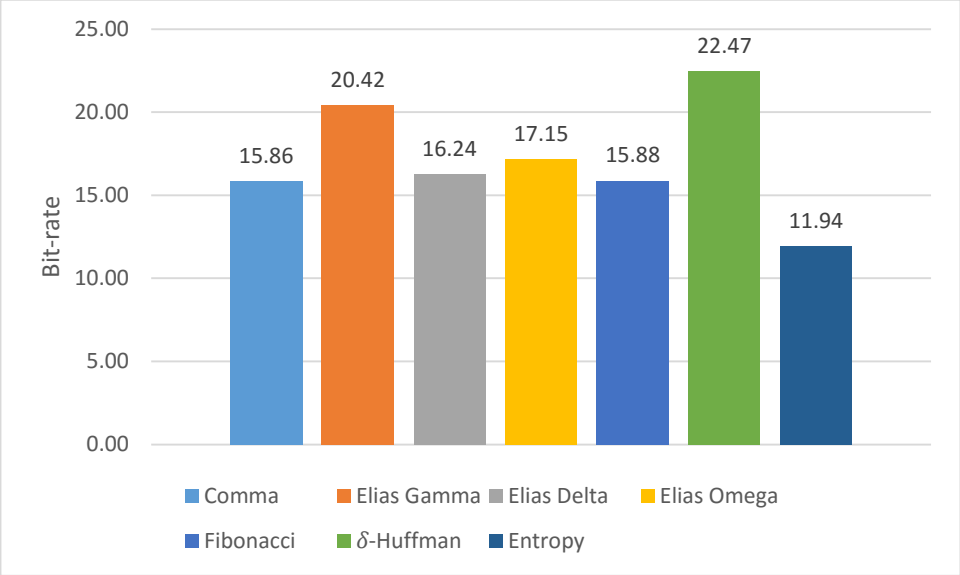


Figure 5.2(c): The bit-rate of the compression algorithms – Bollywood

The R-squared value of the "**Bollywood**" histogram is 0.1354 and it is not a good fit. The equation of "**Bollywood**" do not have a close match with the equations of GPDF.

Figure 5.3(a) provides the histogram of "Film". Figures 5.3(b) – 5.3(c) provide the bit-rate of the Golomb coding and compression algorithms such as Comma code, Elias Gamma code, Elias Delta code, Elias Omega code, Fibonacci code, and δ-Huffman code.

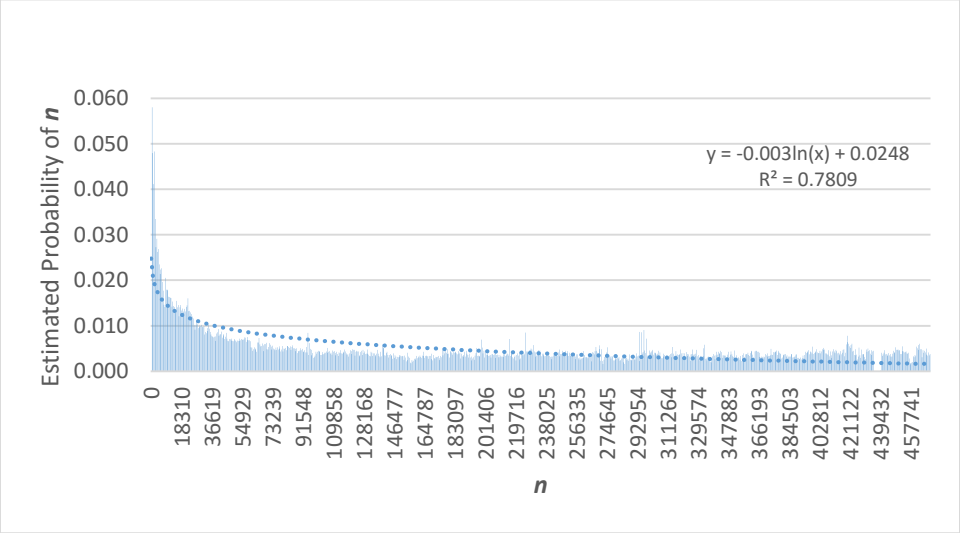


Figure 5.3(a): The histogram of Film

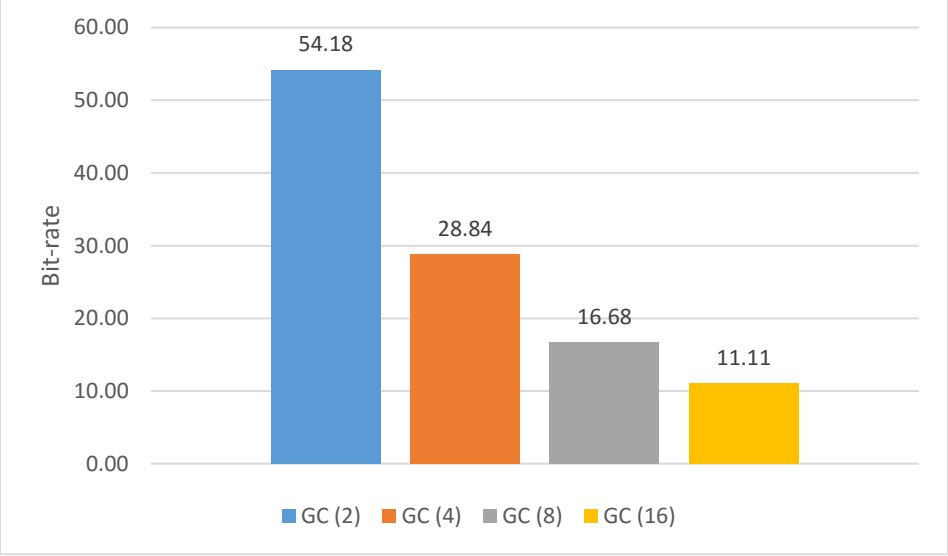


Figure 5.3(b): The bit-rate of the Golomb coding – Film

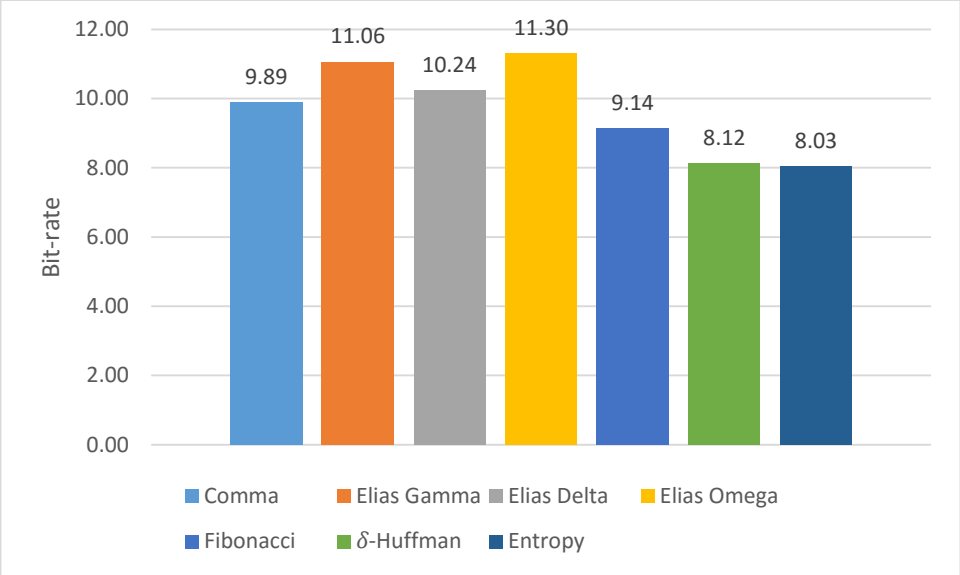


Figure 5.3(c): The bit-rate of the compression algorithms – Film

The R-squared value of the "**Film**" histogram is 0.7809 and it is a good fit. The equation of "**Film**" histogram, which is $y = -0.003ln(x) + 0.0248$ is close to the equation of GPDF (0.01) which is $y = -0.016ln(x) + 0.0699$.

Comparing the equations of the histograms, we can expect that the data-sets would have δ-Huffman code averages within a close margin or range. The calculated average of the δ-Huffman code of "Film" is 8.12 bits per integer. The calculated average of the δ-Huffman codeof GPDF (0.01) is 8.86 bits per integer. In this case, we can observe that the averages vary by an average of 0.74 bits per integer.

Figure 5.4(a) provides the histogram of "Grei". Figures 5.4(b) – 5.4(c) provide the bit-rate of the Golomb coding and compression algorithms such as Comma code, Elias Gamma code, Elias Delta code, Elias Omega code, Fibonacci code, and δ-Huffman code.

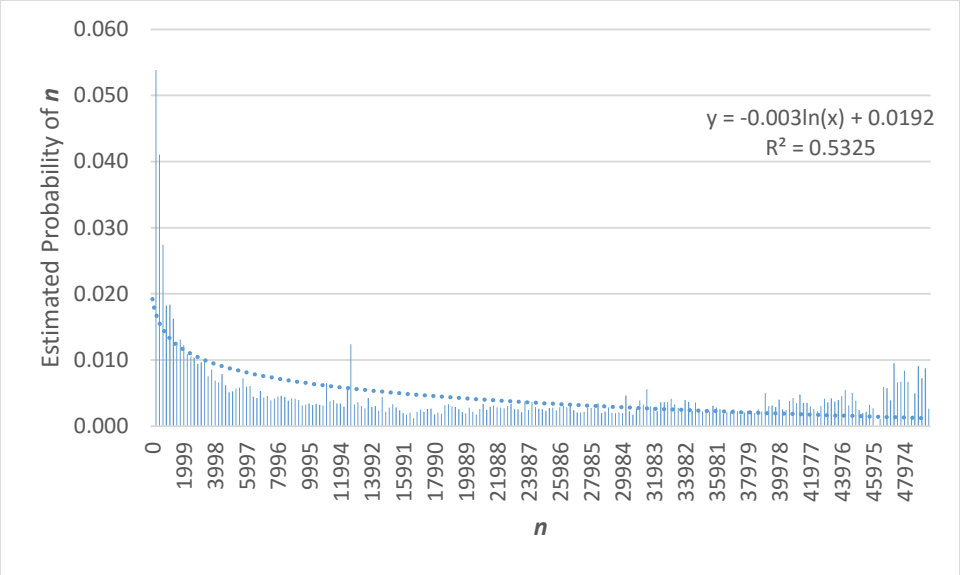


Figure 5.4(a): The histogram of Grei

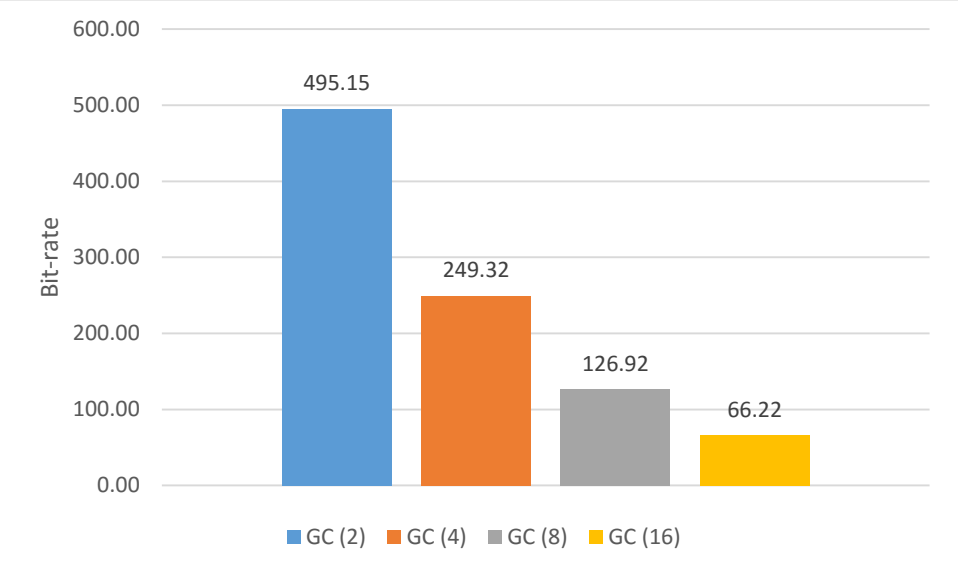


Figure 5.4(b): The bit-rate of the Golomb coding – Grei

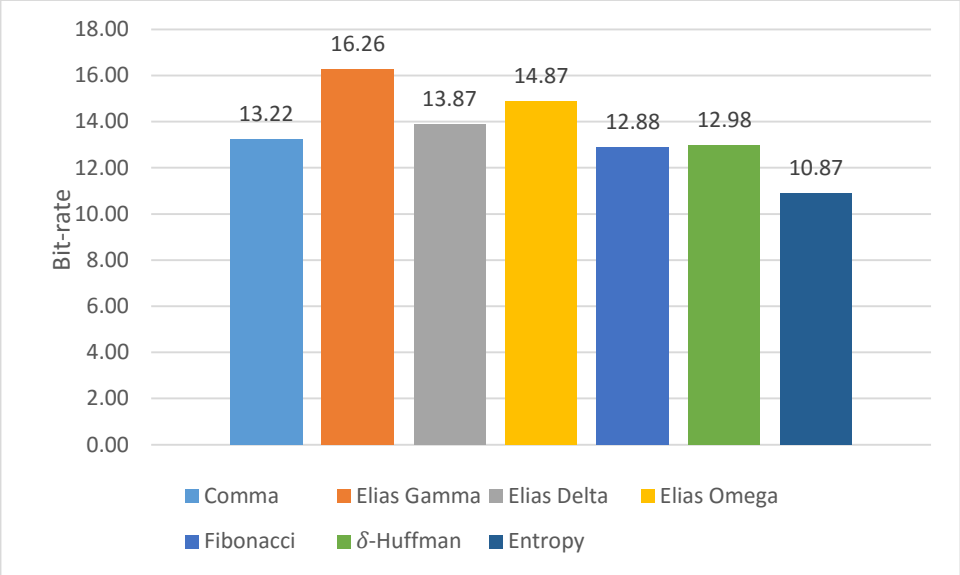


Figure 5.4(c): The bit-rate of the compression algorithms – Grei

The R-squared value of the "**Grei**" histogram is 0.5325 and it is a medium good fit. The equation of "**Grei**" histogram is $y = -0.003ln(x) + 0.0192$. The equation of "**Grei**" do not have a close match with the equations of GPDF.

Figure 5.5(a) provides the histogram of "India". Figures 5.5(b) – 5.5(c) provide the bit-rate of the Golomb coding and compression algorithms such as Comma code, Elias Gamma code, Elias Delta code, Elias Omega code, Fibonacci code, and δ-Huffman code.

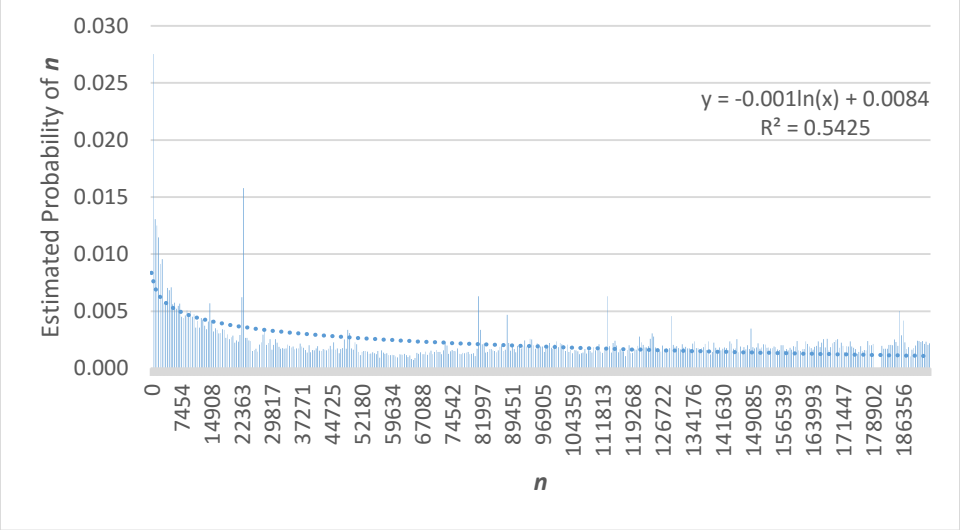


Figure 5.5(a): The histogram of India

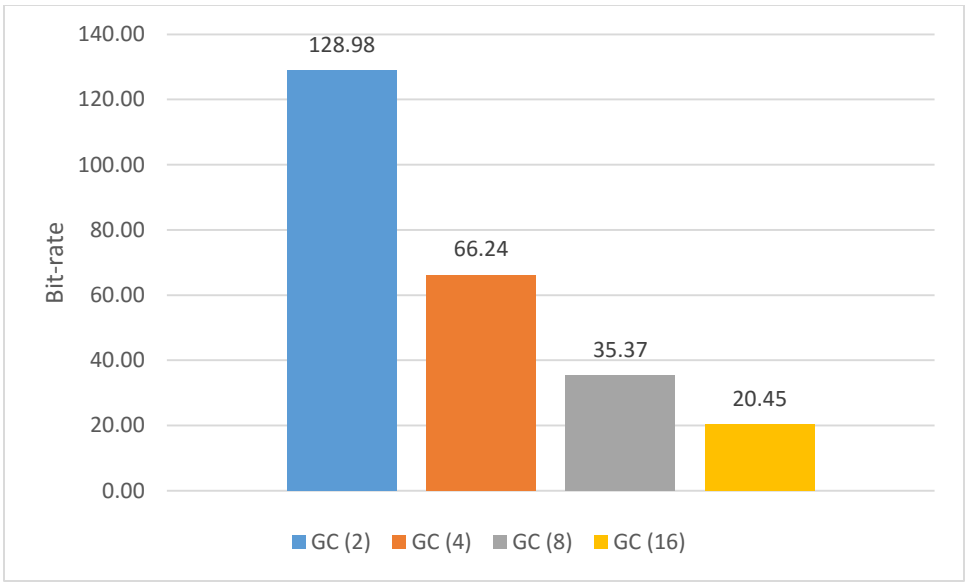


Figure 5.5(b): The bit-rate of the Golomb coding – India

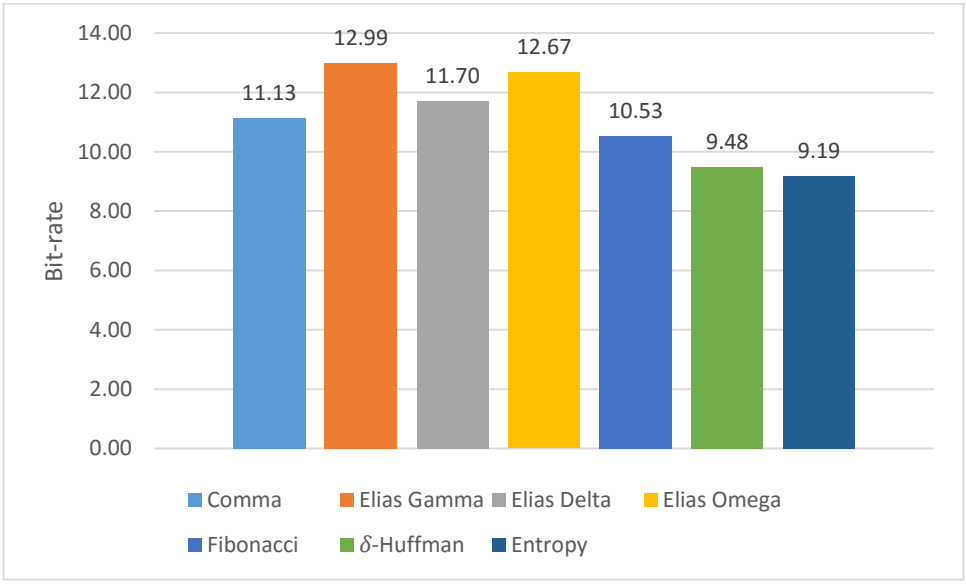


Figure 5.5(c): The bit-rate of the compression algorithms – India

The R-squared value of the "**India**" histogram is 0.5425 and it is a medium good fit. The equation of the "**India**" histogram, which is $y = -0.001ln(x) + 0.0084$ is close to the equation of GPDF (0.01) which is $y = -0.016ln(x) + 0.0699$.

Comparing the above equations, we can expect that the data-sets would have δ-Huffman code averages within a close margin or range. The calculated average of the δ-

Huffman code of "**India**" is 9.48 bits per integer. The calculated average of the δ-Huffman code of GPDF (0.01) is 8.86 bits per integer. In this case, we can observe that the averages differ by 0.62 bits per integer.

Figure 5.6(a) provides the histogram of "Rousei". Figures 5.6(b) – 5.6(c) provide the bit-rate of the Golomb coding and compression algorithms such as Comma code, Elias Gamma code, Elias Delta code, Elias Omega code, Fibonacci code, and δ-Huffman code.

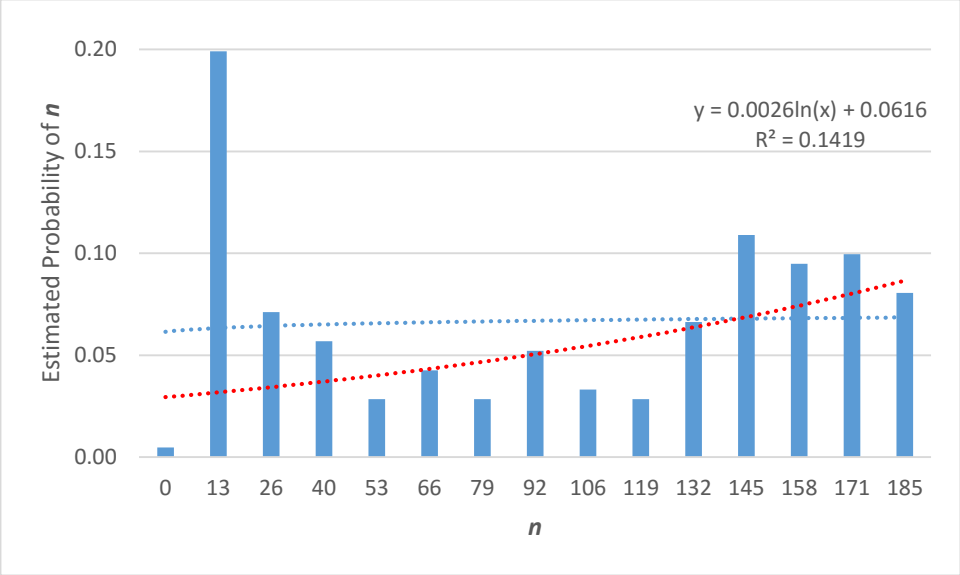


Figure 5.6(a): The histogram of Rousei

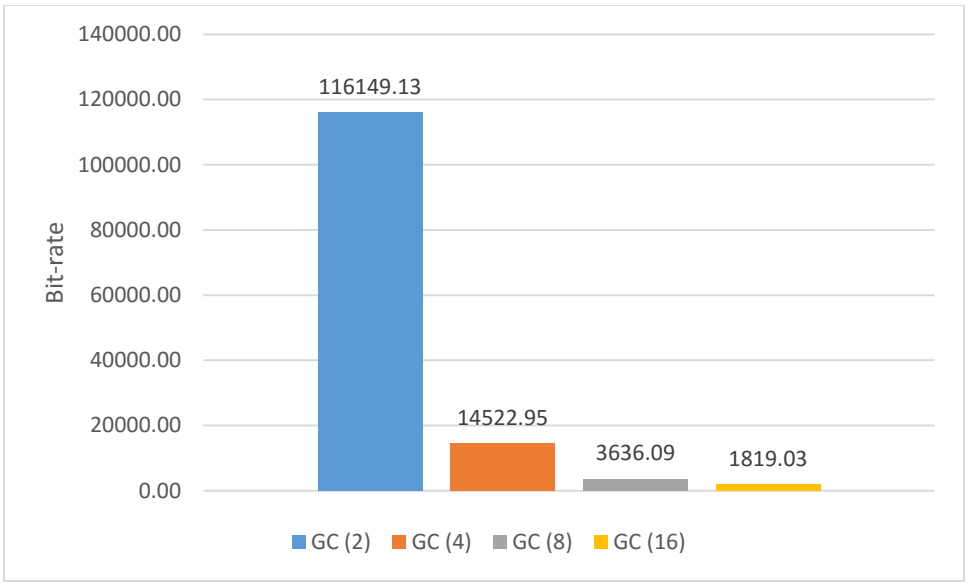


Figure 5.6(b): The bit-rate of the Golomb coding – Rousei

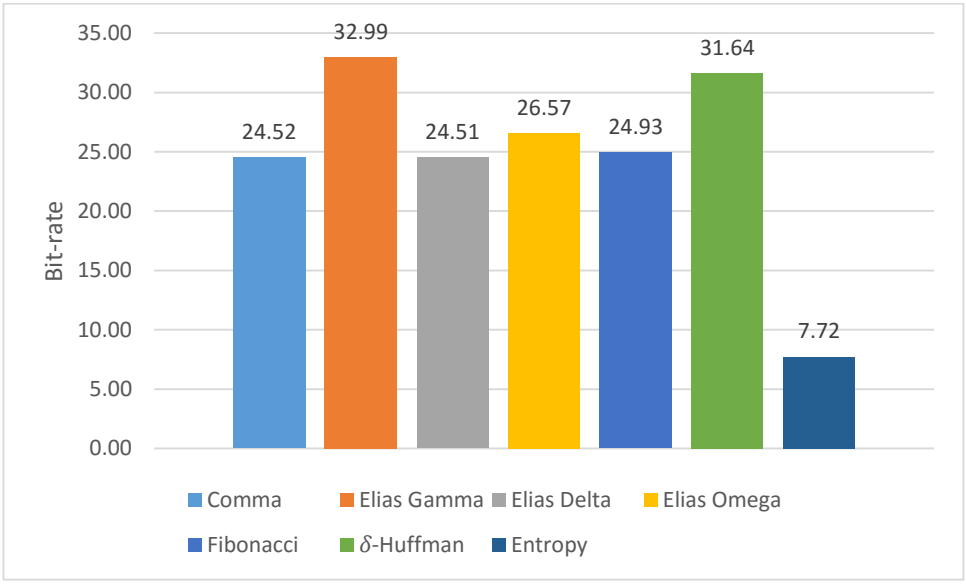


Figure 5.6(c): The bit-rate of the compression algorithms – Rousei

The R-squared value of the "**Rousei**" histogram is 0.00001 and it is not a good fit. The R-squared value (the exponential curve – red color) of the "**Rousei**" histogram is 0.1419, which is not a good fit either, but it is better than the logarithmic curve (blue color which is 0.000012). Hence, the equation of "**Rousei**" do not have a close match with the equations of GPDF.

Figure 5.7(a) provides the histogram of "State". Figures 5.7(b) – 5.7(c) provide the bit-rate of the Golomb coding and compression algorithms such as Comma code, Elias Gamma code, Elias Delta code, Elias Omega code, Fibonacci code, and δ-Huffman code.

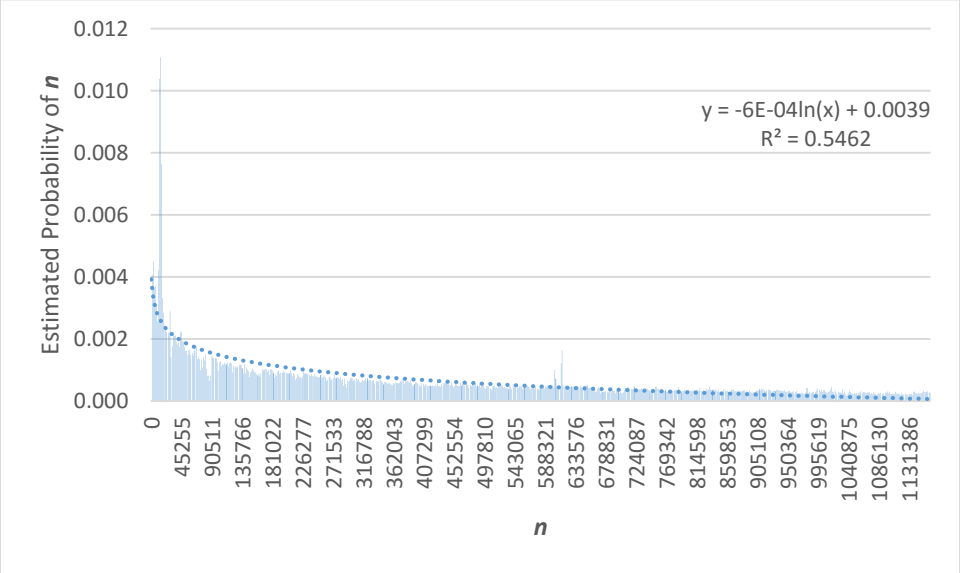


Figure 5.7(a): The histogram of State

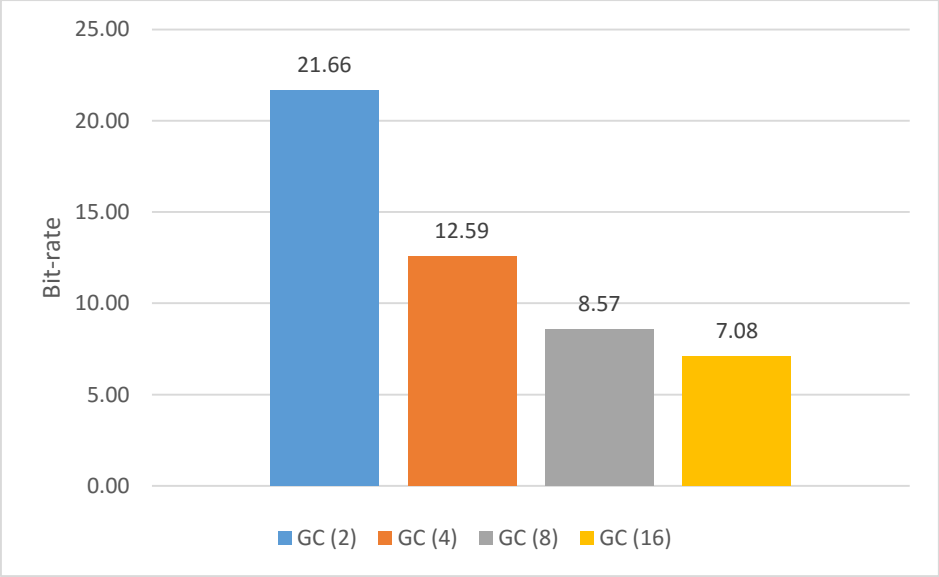


Figure 5.7(b): The bit-rate of the Golomb coding - State

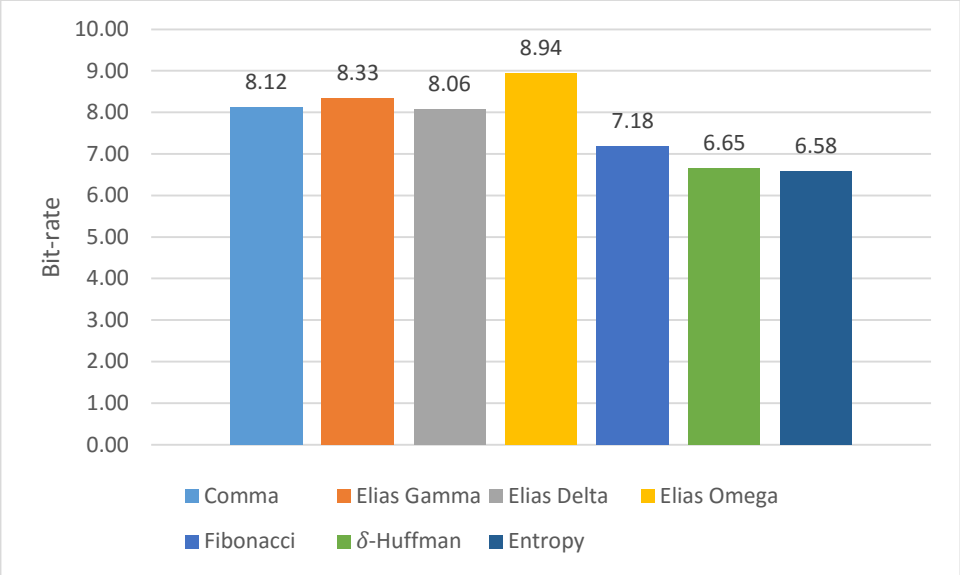


Figure 5.7(c): The bit-rate of the compression algorithms - State

The R-squared value of the "**State**" histogram is 0.5462 and it is a better fit. The equation of the "**State**" histogram, which is $y = -6E-04ln(x) + 0.0039$ is close to the equation of GPDF (0.01) which is $y = -0.016ln(x) + 0.0699$.

Comparing the equations, we expect that the data-sets would have a δ-Huffman code averages within a close margin or range. The calculated average of the δ-Huffman code of "**State**" is 6.65 bits per integer. The calculated average of the δ-Huffman code of GPDF (0.01) is 8.86 bits per integer. In this case, we can observe that the averages vary by an average of 2.21 bits per integer.

Figure 5.8(a) provides the histogram of "Stephen". Figures 5.8(b) – 5.8(c) provide the bit-rate of the Golomb coding and compression algorithms such as Comma code, Elias Gamma code, Elias Delta code, Elias Omega code, Fibonacci code, and δ-Huffman code.

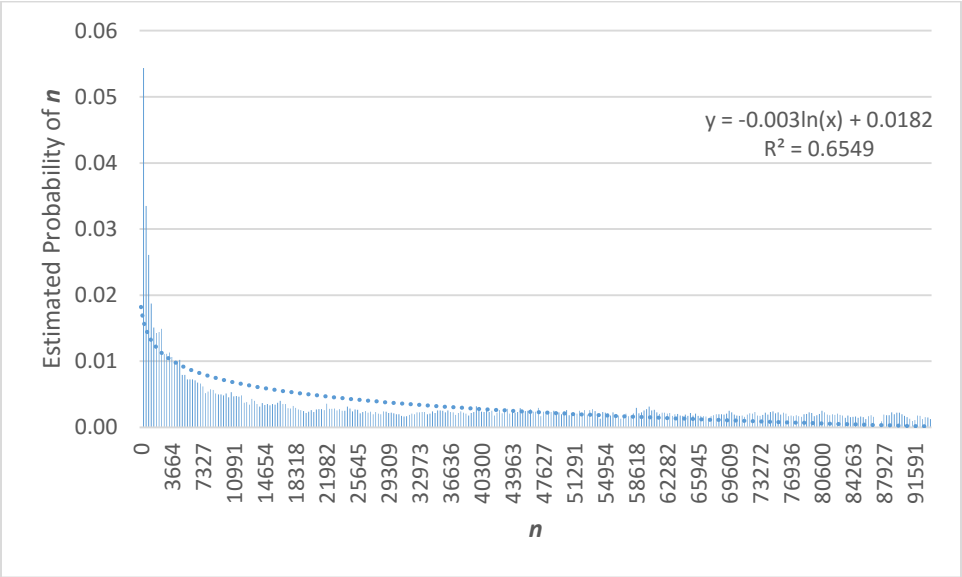


Figure 5.8(a): The histogram of Stephen

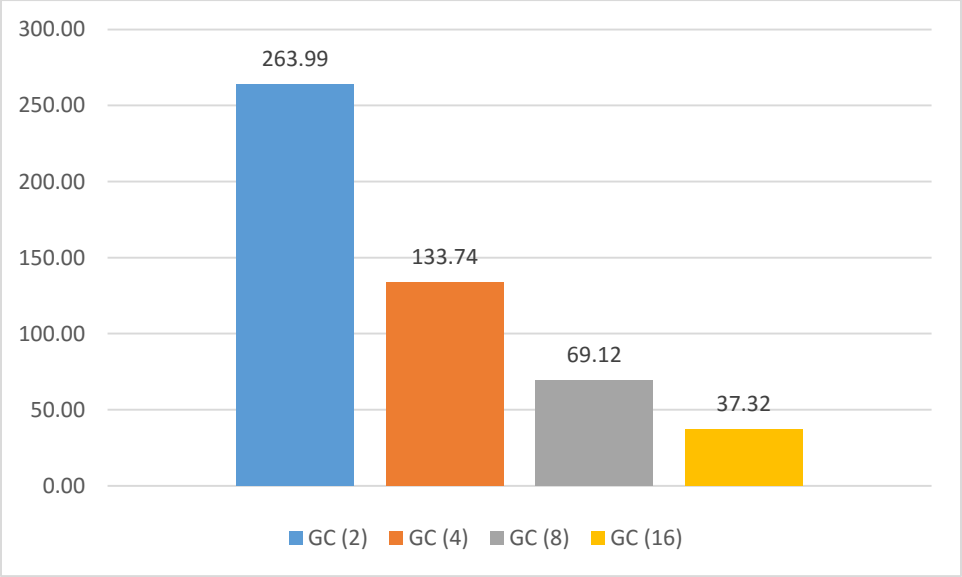


Figure 5.8(b): The bit-rate of the Golomb coding - Stephen

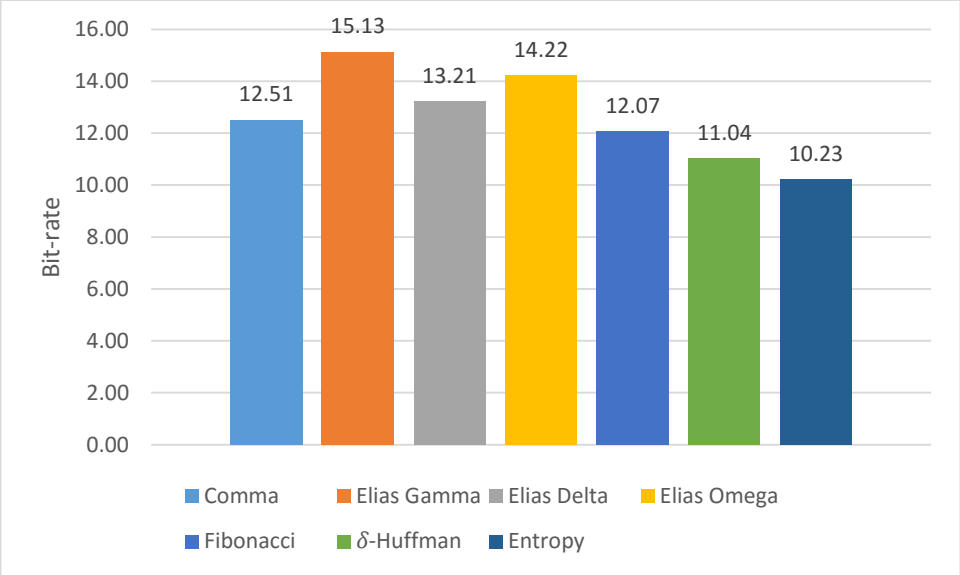


Figure 5.8(c): The bit-rate of the compression algorithms - Stephen

The R-squared value of the "**Stephen**" histogram is 0.6549 and it is a good fit. The equation of the "**Stephen**" histogram, which is $y = -0.003ln(x) + 0.0182$ is close to the equation of GPDF (0.01) which is $y = -0.016ln(x) + 0.0699$.

Comparing the equations, we expect that the data-sets would have an δ-Huffman code averages within a close margin or range. The calculated average of the δ-Huffman code of "**Stephen**" is 11.04 bits per integer. The average of the δ-Huffman code of GPDF (0.01) is 8.86 bits per integer.

Figure 5.9(a) provides the histogram of "Walker". Figures 5.9(b) – 5.9(c) provide the bit-rate of the Golomb coding and compression algorithms such as Comma code, Elias Gamma code, Elias Delta code, Elias Omega code, Fibonacci code, and δ-Huffman code.

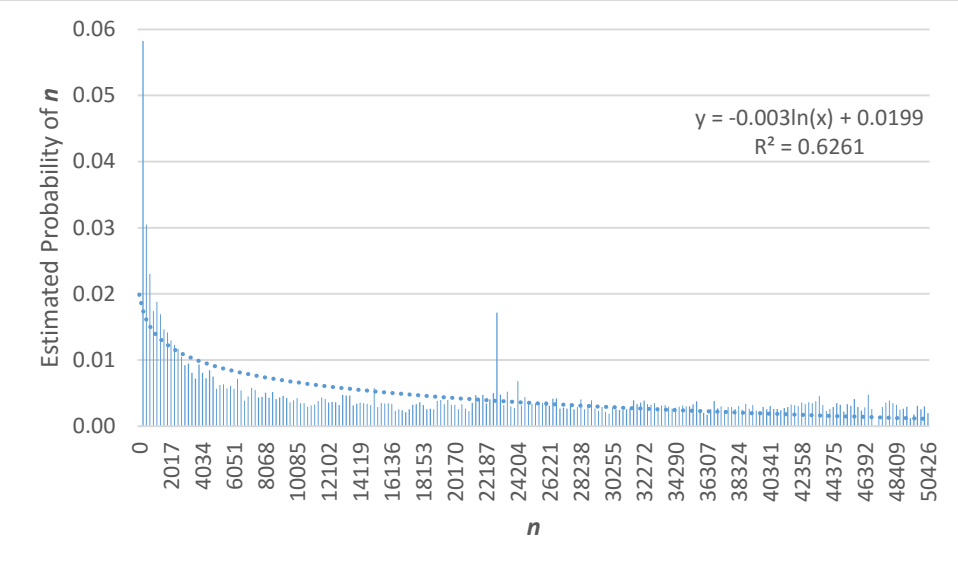


Figure 5.9(a): The histogram of Walker

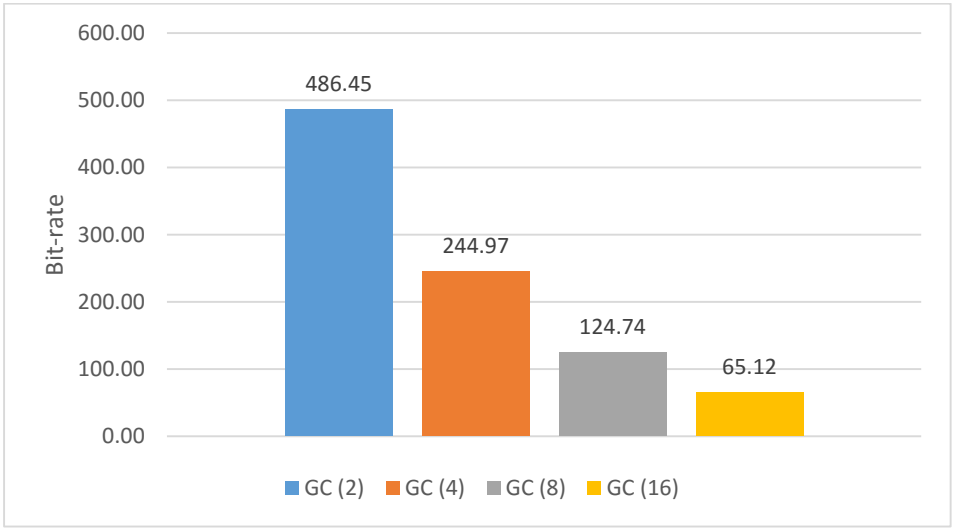


Figure 5.9(b): The bit-rate of the Golomb coding – Walker

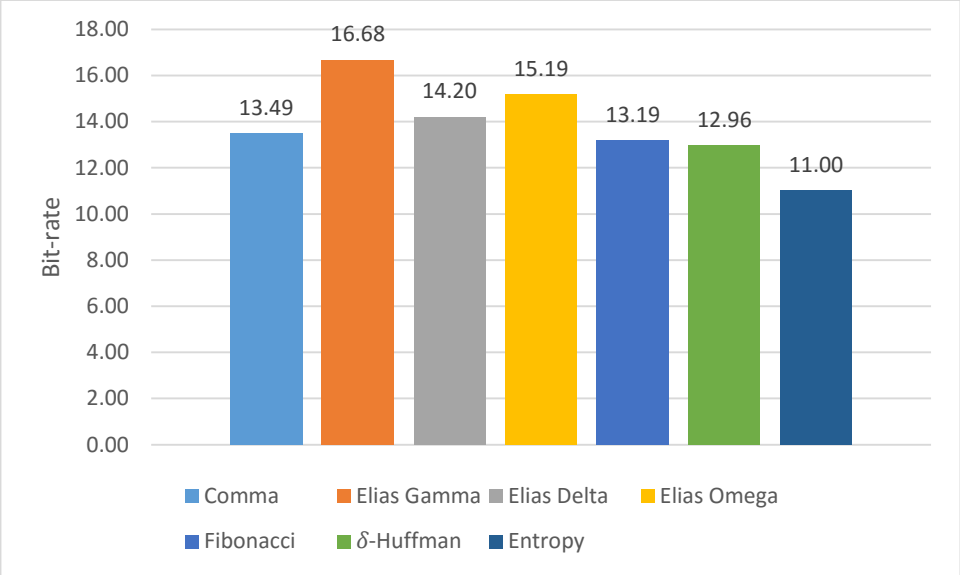


Figure 5.9(c): The bit-rate of the compression algorithms – Walker

The R-squared value of the "**Walker**" histogram is 0.6261 and it is a good fit. The equation of the "**Walker**" histogram, which is $y = -0.003ln(x) + 0.0199$ is close to the equation of GPDF (0.01) which is $y = -0.016ln(x) + 0.0699$.

Comparing the equations, we expect that the data-sets would have an δ-Huffman code averages within a close margin or range. The calculated average of the δ-Huffman code of "**Walker**" is 12.96 bits per integer. The calculated average of the δ-Huffman code of GPDF (0.01) is 8.86 bits per integer. In this case, we can observe that the averages differ by 4.1 bits per integer.

Figure 5.10(a) provides the histogram of "War". Figures 5.10(b) – 5.10(c) provide the bit-rate of the Golomb coding and compression algorithms such as Comma code, Elias Gamma code, Elias Delta code, Elias Omega code, Fibonacci code, and δ-Huffman code.

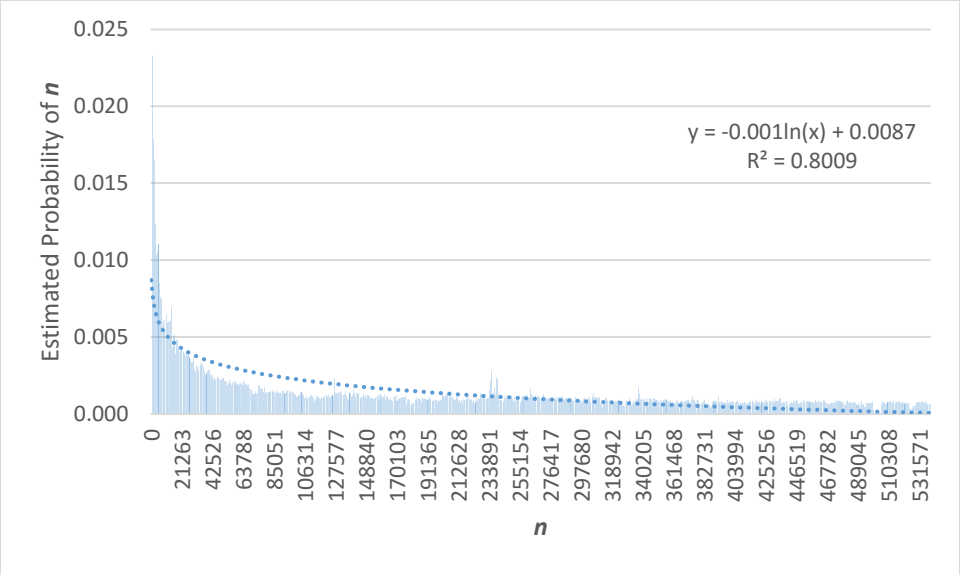


Figure 5.10(a): The histogram of War

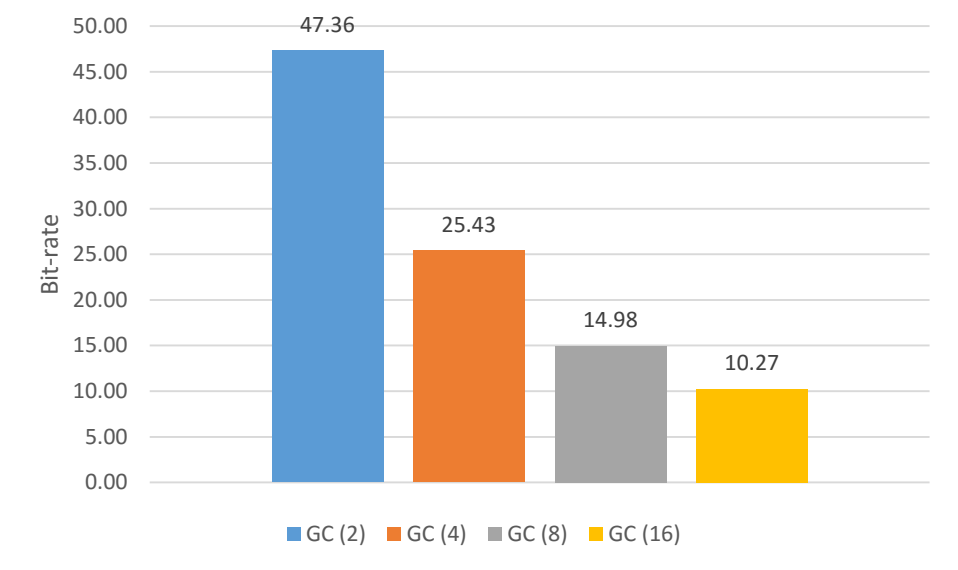


Figure 5.10(b): The bit-rate of the Golomb coding – War

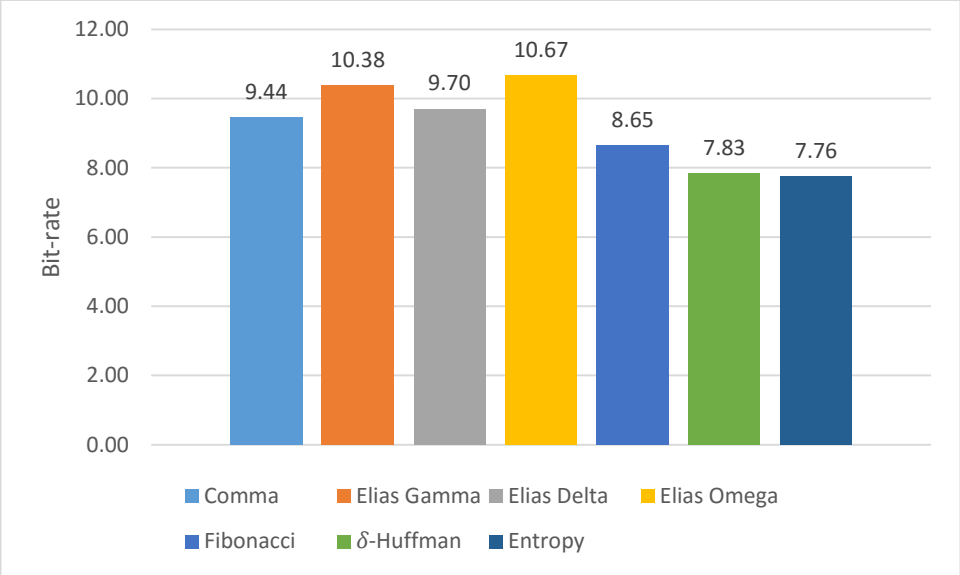


Figure 5.10(c): The bit-rate of the compression algorithms – War

The R-squared value of the "**War**" histogram is 0.8009 and it is the best fit. The equation of the "**War**" histogram, which is $y = -0.001ln(x) + 0.0087$ is close to the equation of GPDF (0.01) which is $y = -0.016ln(x) + 0.0699$.

Comparing the above equations, we can expect that the data-sets would have an δ-Huffman code averages within a close margin or range. The calculated average of the δ-Huffman code of "**War**" is 7.83 bits per integer. The calculated average of the δ-Huffman code of GPDF (0.01) is 8.86 bits per integer. In this case, we can observe that they vary by an average of 1.03 bits per integer.

Figure 5.11(a) provides the histogram of "West". Figures 5.11(b) – 5.11(c) provide the bit-rate of the Golomb coding and compression algorithms such as Comma code, Elias Gamma code, Elias Delta code, Elias Omega code, Fibonacci code, and δ-Huffman code.

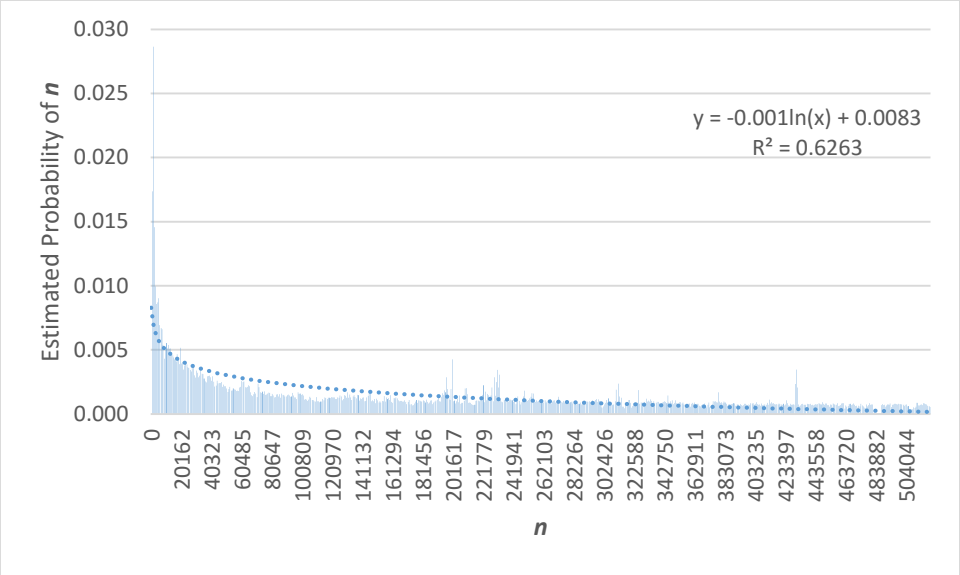


Figure 5.11(a): The histogram of West

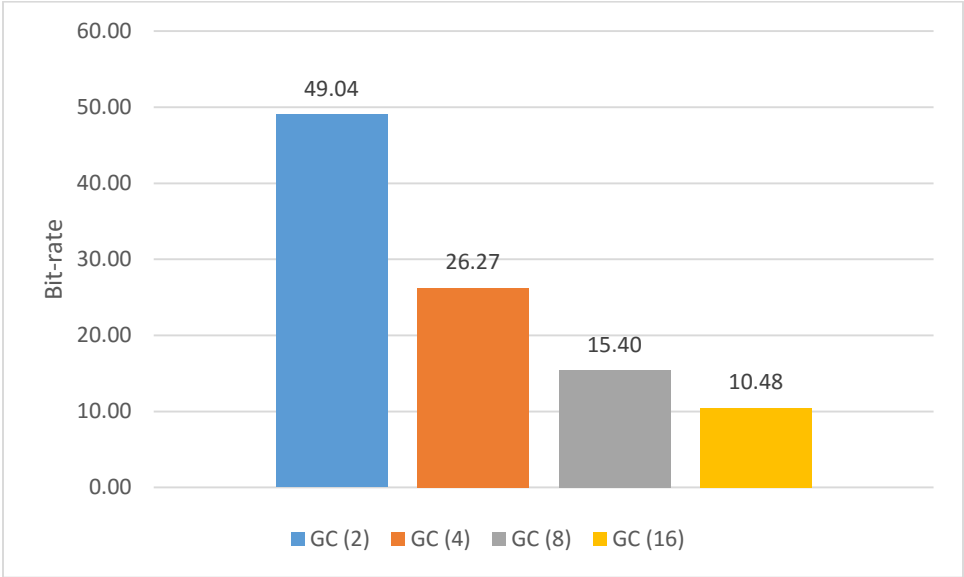


Figure 5.11(b): The bit-rate of the Golomb coding – West

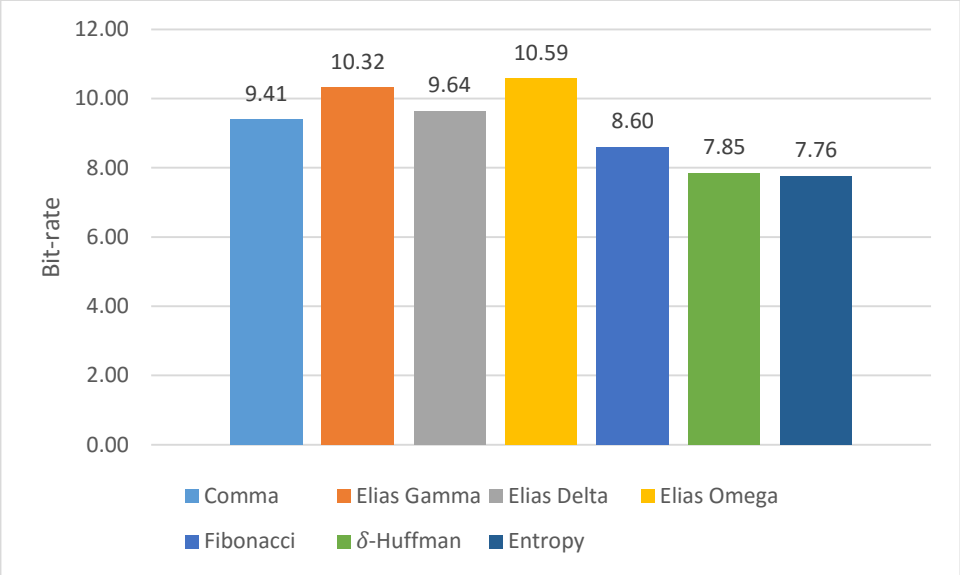


Figure 5.11(c): The bit-rate of the compression algorithms – West

The R-squared value of the "**West**" histogram is 0.6263 and it is a better fit. The equation of the "**West**" histogram, which is $y = -0.001ln(x) + 0.0083$ is close to the equation of GPDF (0.01) which is $y = -0.016ln(x) + 0.0699$.

Comparing the above equations, we can expect that the data-sets would have an δ-Huffman code averages within a close margin or range. The calculated average of the δ-Huffman code of "**West**" is 7.85 bits per integer. The calculated average of the δ-Huffman code of GPDF (0.01) is 8.86 bits per integer. In this case, we can observe that they vary by an average of 1.11 bits per integer.

Figure 5.12(a) provides the histogram of "World". Figures 5.12(b) – 5.12(c) provide the bit-rate of the Golomb coding and compression algorithms such as Comma code, Elias Gamma code, Elias Delta code, Elias Omega code, Fibonacci code, and δ-Huffman code.

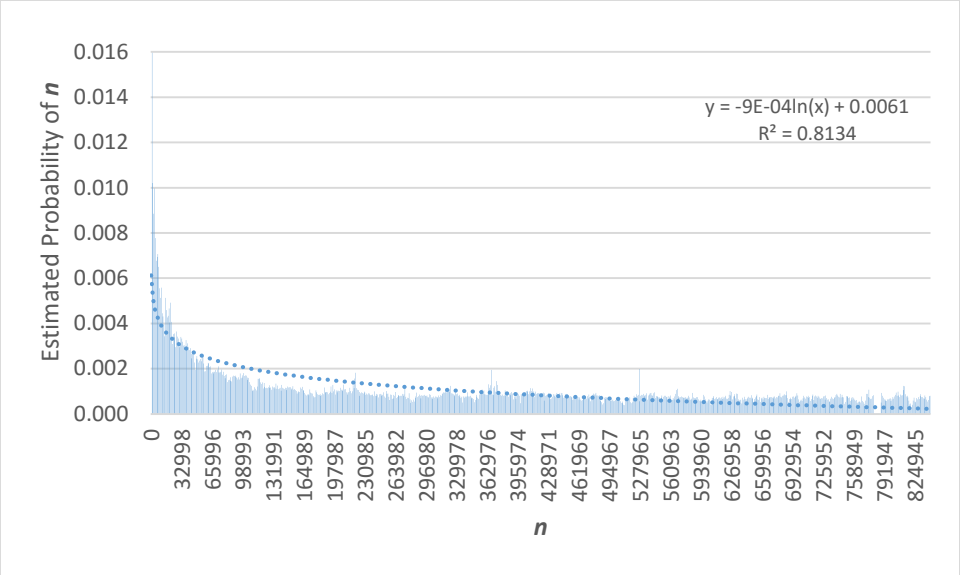


Figure 5.12(a): The histogram of World

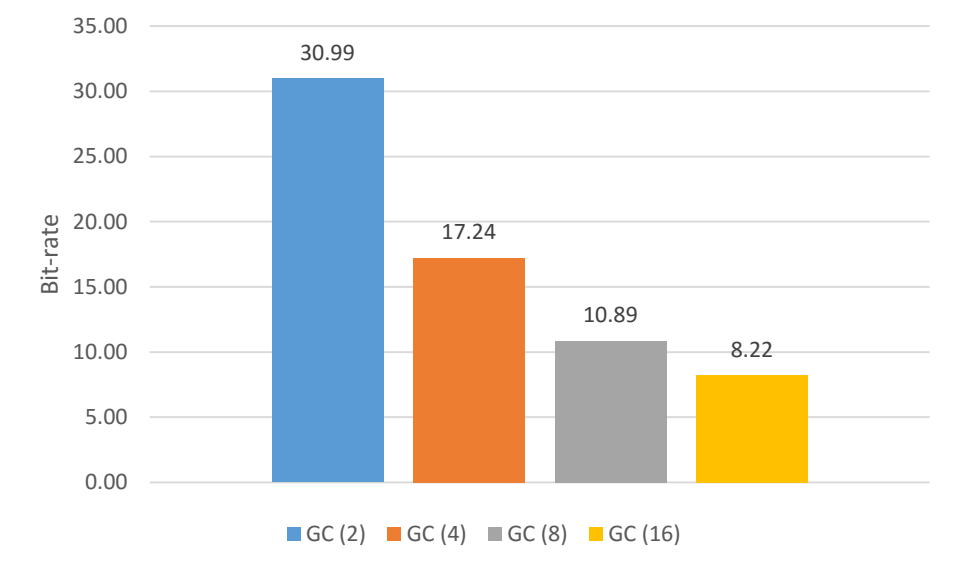


Figure 5.12(b): The bit-rate of the Golomb coding – World

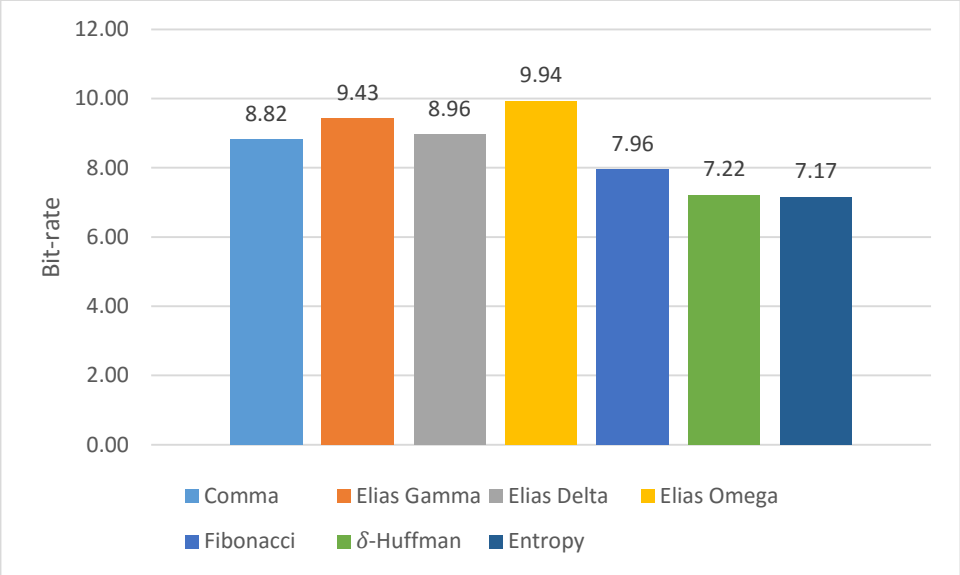


Figure 5.12(c): The bit-rate of the compression algorithms – World

The R-squared value of the "**World**" histogram is 0.8134 and it is the best fit. The equation of the "**World**", which is $y = -9E-04ln(x) + 0.0061$ is close to the equation of GPDF (0.01) which is $y = -0.016ln(x) + 0.0699$.

Comparing the above equations, we expect that the data-sets would have a δ-Huffman code averages within a close margin or range. The calculated average of the δ-Huffman code of "**World**" is 7.22 bits per integer. The calculated average of the δ-Huffman code of GPDF (0.01) is 8.86 bits per integer. In this case, we observe that they vary by an average of 1.64 bits per integer.

Figure 5.13(a) provides the histogram of "Facebook". Figures 5.13(b) – 5.13(c) provide the bit-rate of the Golomb coding and compression algorithms such as Comma code, Elias Gamma code, Elias Delta code, Elias Omega code, Fibonacci code, and δ-Huffman code.

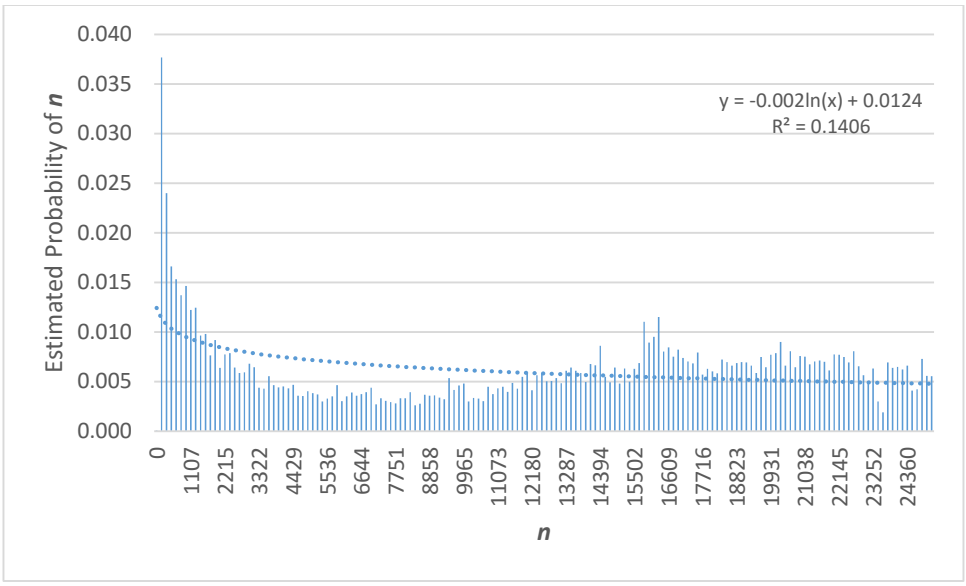


Figure 5.13(a): The histogram of Facebook

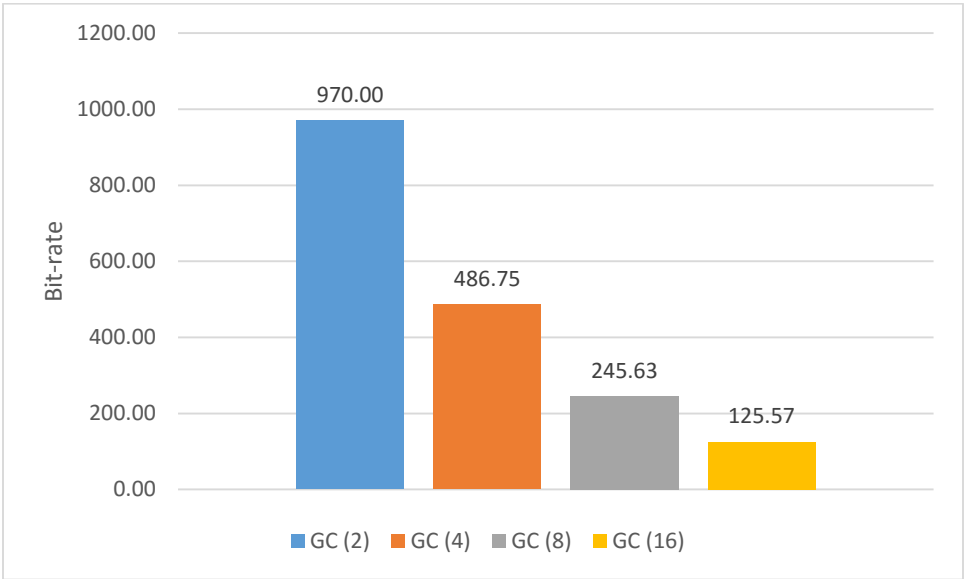


Figure 5.13(b): The bit-rate of the Golomb coding – Facebook

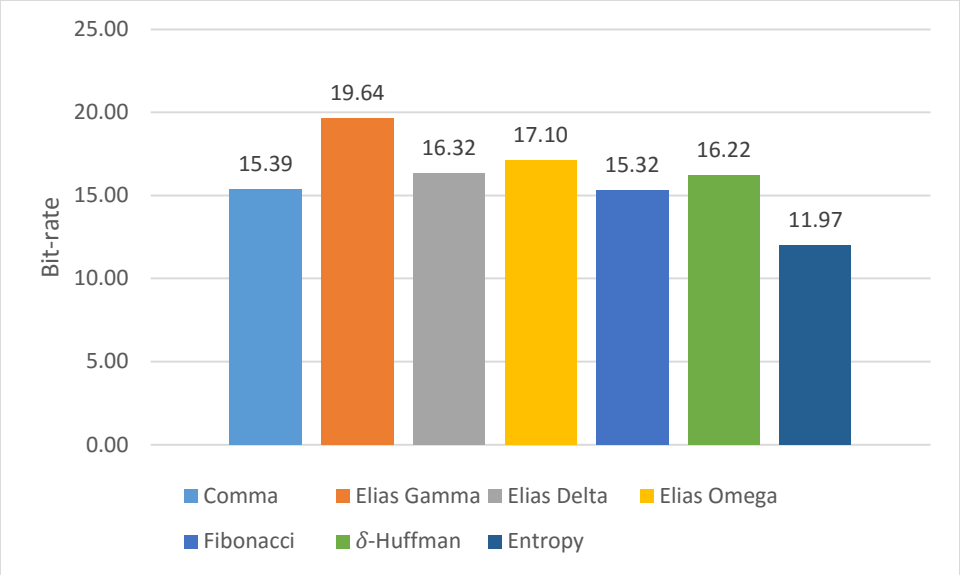


Figure 5.13(c): The bit-rate of the compression algorithms – Facebook

The R-squared value of the "**Facebook**" histogram is 0.1406 and it is not a good fit. The equation of "**Facebook**" do not have a close match with the equations of GPDF.

Figure 5.14(a) provides the histogram of "Iraq". Figures 5.14(b) – 5.14(c) provide the bit-rate of the Golomb coding and compression algorithms such as Comma code, Elias Gamma code, Elias Delta code, Elias Omega code, Fibonacci code, and δ-Huffman code.

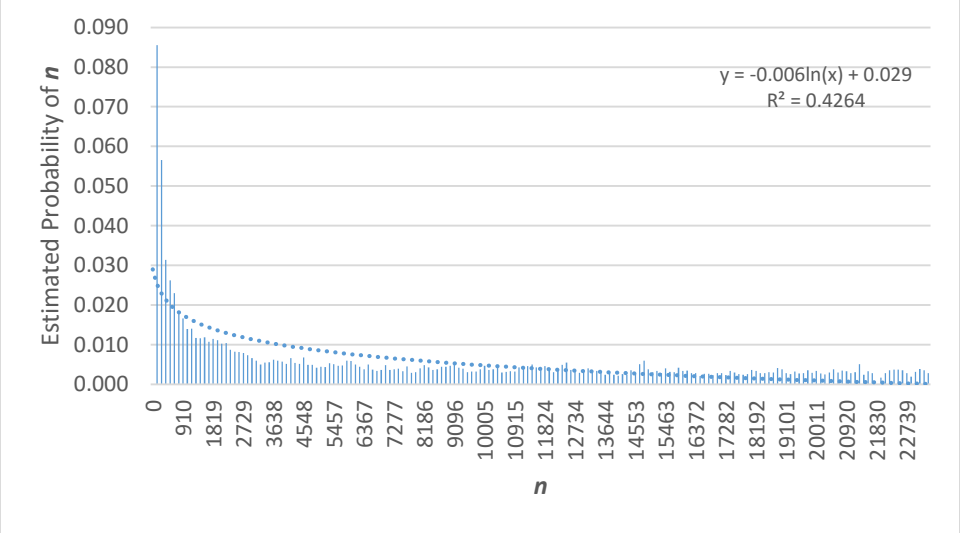


Figure 5.14(a): The histogram of Iraq

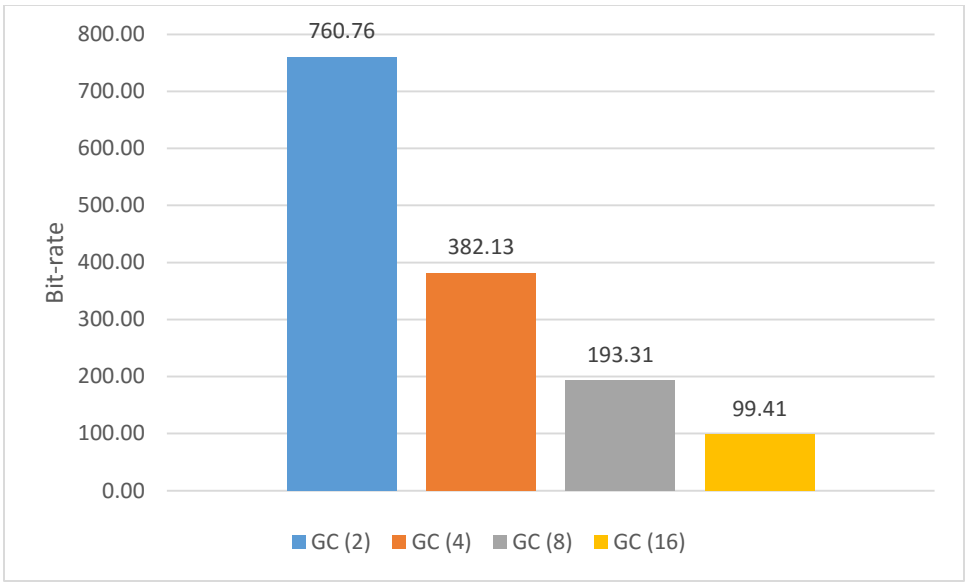


Figure 5.14(b): The bit-rate of the Golomb coding – Iraq

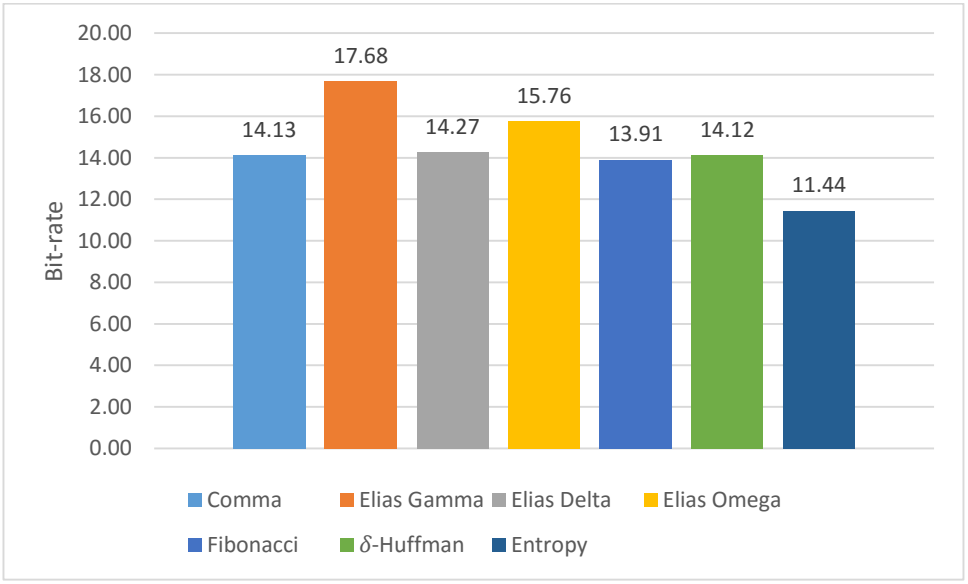


Figure 5.14(c): The bit-rate of the compression algorithms – Iraq

The R-squared value of the "**Iraq**" histogram is 0.4264 and it is not a good fit. The equation of "**Iraq**" do not have a close match with the equations of GPDF.

Figure 5.15(a) provides the histogram of "Obama". Figures 5.15(b) – 5.15(c) provide the bit-rate of the Golomb coding and compression algorithms such as Comma code, Elias Gamma code, Elias Delta code, Elias Omega code, Fibonacci code, and δ-Huffman code.

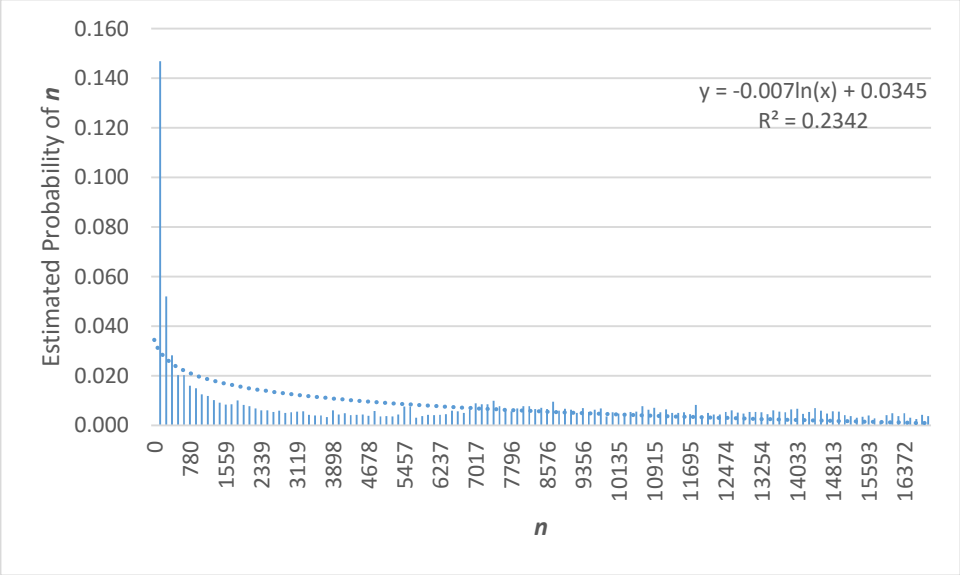


Figure 5.15(a): The histogram of Obama

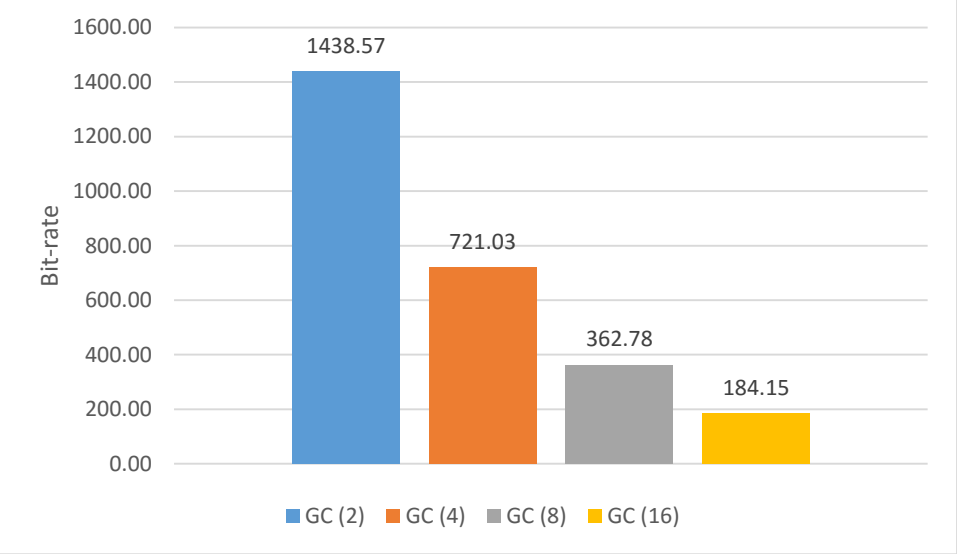


Figure 5.15(b): The bit-rate of the Golomb coding – Obama

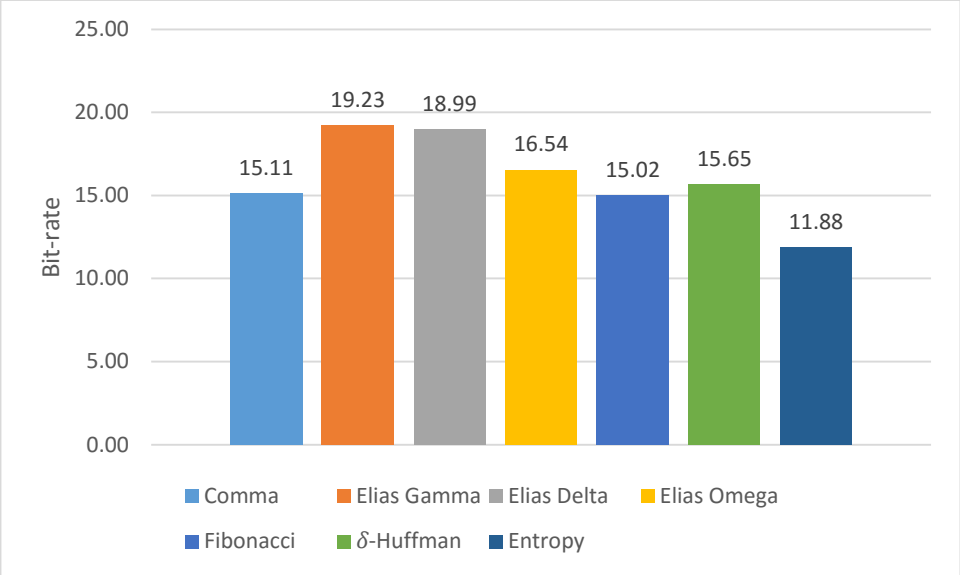


Figure 5.15(c): The bit-rate of the compression algorithms – Obama

The R-squared value of the "**Obama**" histogram is 0.2342 and it is not a good fit. The equation of "**Obama**" do not have a close match with the equations of GPDF.

From all of the above experiments on the gaps of sorted inverted lists, we can observe that except in 1 or 2 cases (specifically with the lowest data-set, Rousei with an input of 211 integers), for all the data-sets which are having more than 50, 000 entries δ-Huffman code provides low bit-rate which is relatively close to the entropy. The reason for the best performance of δ-Huffman code compared to other compression techniques is that when the gaps are implemented on the data-sets of sorted inverted lists, the resultant data-set contains quite a lot of repetitions and the gaps occur in closer intervals. Because of the numerous repetitions and the locality of closer intervals, to encode an integer most of the δ-Huffman codes are derived from the Huffman tree (because the repetitions are already updated on the tree), which provides the smallest code word length for encoding the high probability of occurrence of integers.

4.3.6 Experiment 6

In this experiment, we have used the data-sets of gaps of gaps of sorted inverted lists. Along with, additionally we have compared the compression techniques Elias Delta code and δ-Huffman code to the entropy on gaps vs. gaps of gaps. For the differences, we have implemented the sign and magnitude representation (see section 4.2.1). In all of the following graphs in this section, "G" refers to gaps and "GG" refers to gaps of gaps, the left column bar (blue) have provided the results of the compression techniques applied to the data-set of gaps and the right column bar (yellow) have provided the results of the compression techniques applied to the data-set of gaps of gaps.

Below, we have provided the graphs of gaps vs. gaps of gaps of 10 sorted inverted lists. In all of the graphs, we have compared Elias Delta code and δ-Huffman code to the entropy.

Figure 6.1 provides the bit-rate of the compression algorithms Elias Delta code and δ-Huffman code for the gaps and gaps of gaps of inverted list – "2015".

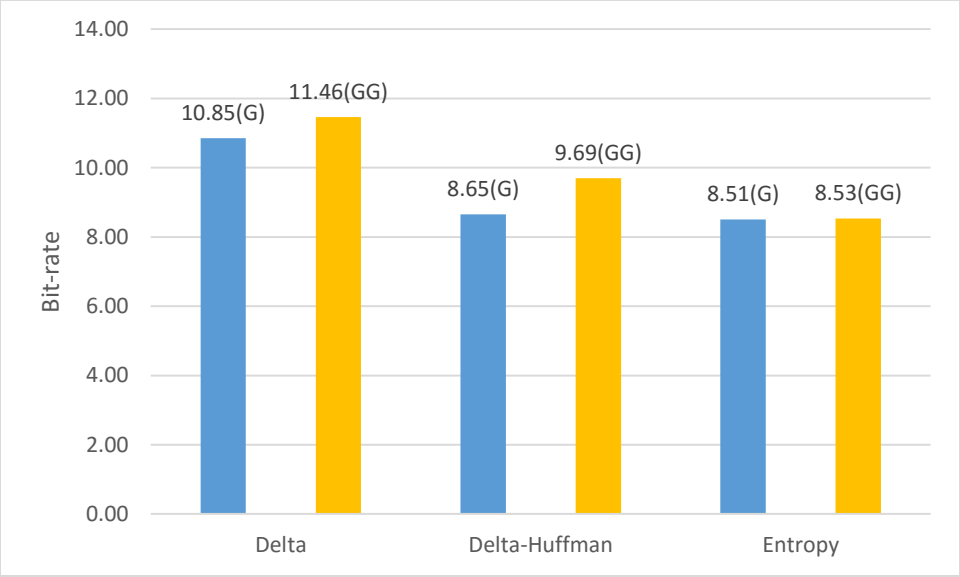


Figure 6.1: The bit-rate of 2015 gaps vs. gaps of gaps

Figure 6.2 provides the bit-rate of the compression algorithms Elias Delta code and δ-Huffman code for the gaps and gaps of gaps of inverted list – "Bollywood".

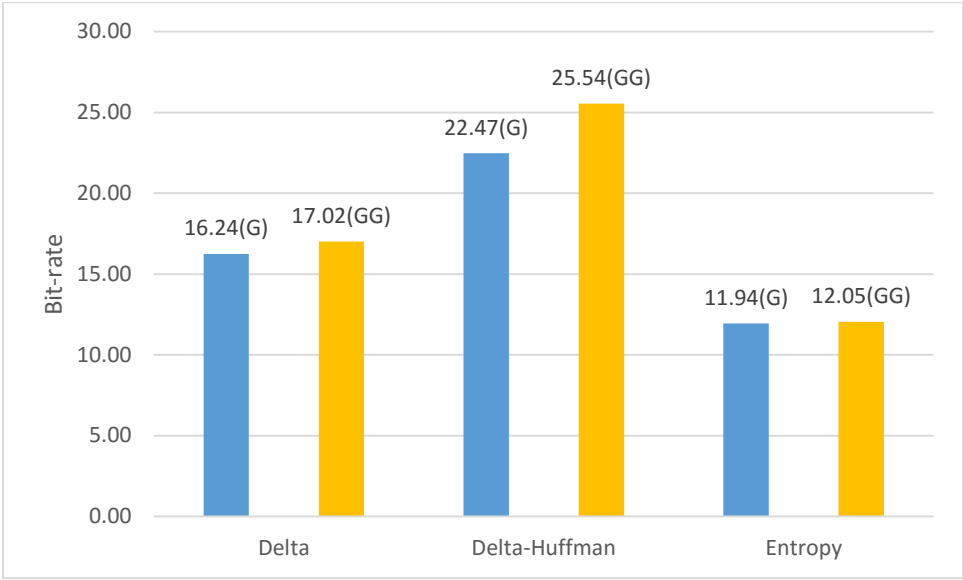


Figure 6.2: The bit-rate of Bollywood gaps vs. gaps of gaps

Figure 6.3 provides the bit-rate of the compression algorithms Elias Delta code and δ-Huffman code for the gaps and gaps of gaps of inverted list – "Film".

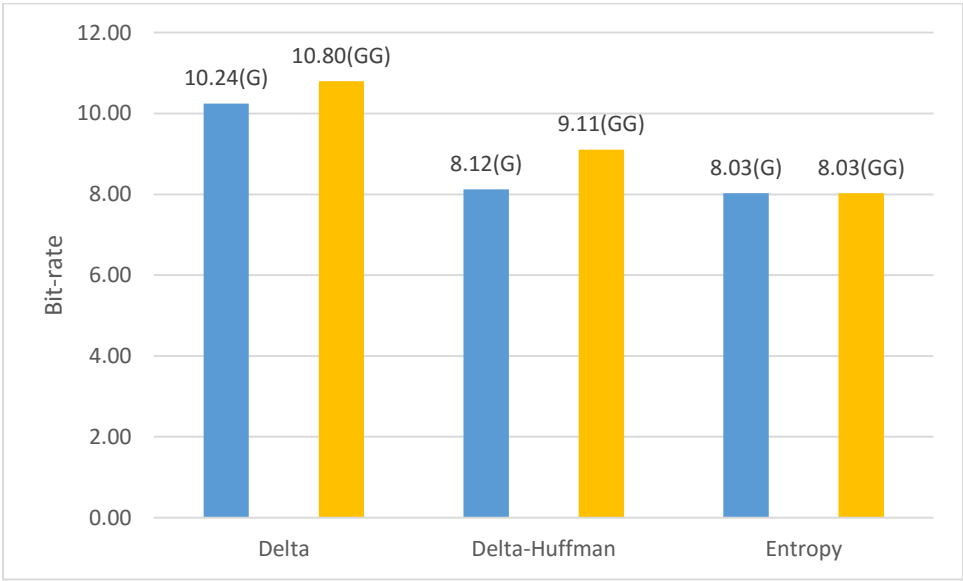


Figure 6.3: The bit-rate of Film gaps vs. gaps of gaps

Figure 6.4 provides the bit-rate of the compression algorithms Elias Delta code and δ-Huffman code for the gaps and gaps of gaps of inverted list – "Grei".

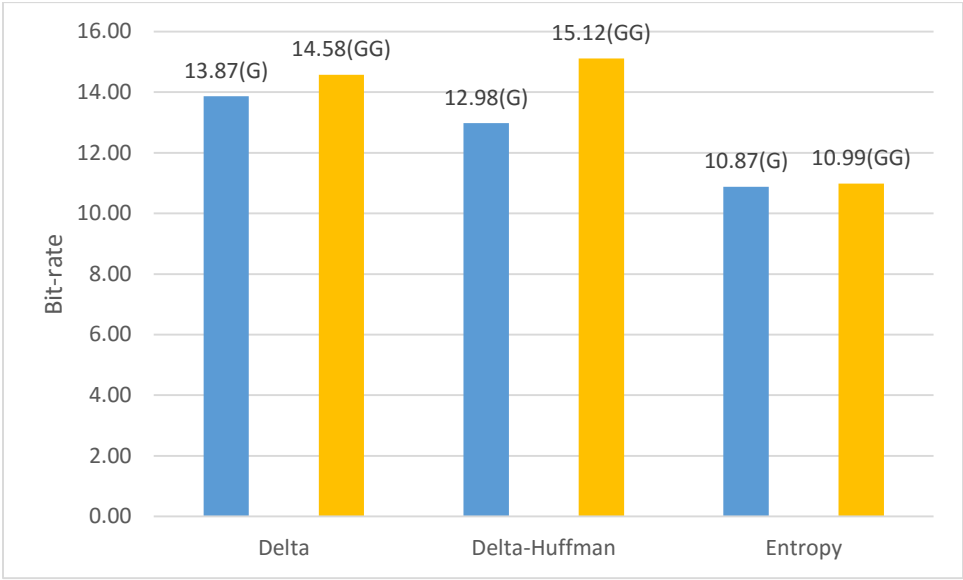


Figure 6.4: The bit-rate of Grei gaps vs. gaps of gaps

Figure 6.5 provides the bit-rate of the compression algorithms Elias Delta code and δ-Huffman code for the gaps and gaps of gaps of inverted list – "India".

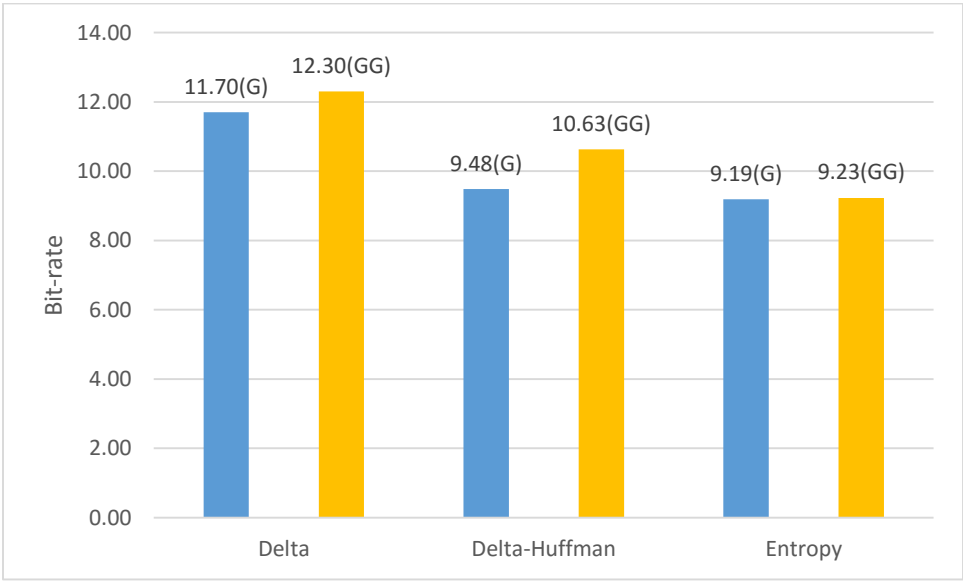


Figure 6.5: The bit-rate of India gaps vs. gaps of gaps

Figure 6.6 provides the bit-rate of the compression algorithms Elias Delta code and δ-Huffman code for the gaps and gaps of gaps of inverted list – "Rousei".

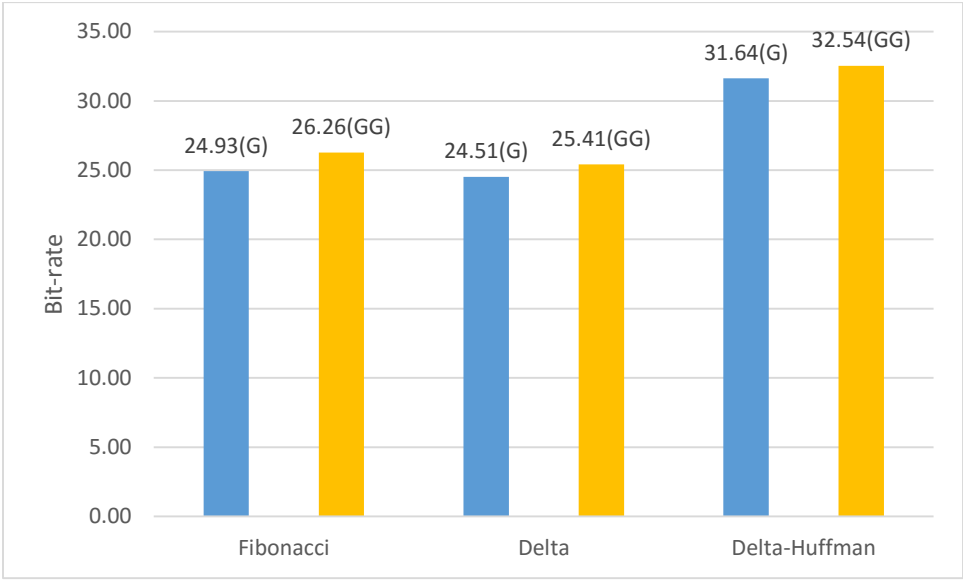


Figure 6.6: The bit-rate of Rousei gaps vs. gaps of gaps

Figure 6.7 provides the bit-rate of the compression algorithms Elias Delta code and δ-Huffman code for the gaps and gaps of gaps of inverted list – "State".

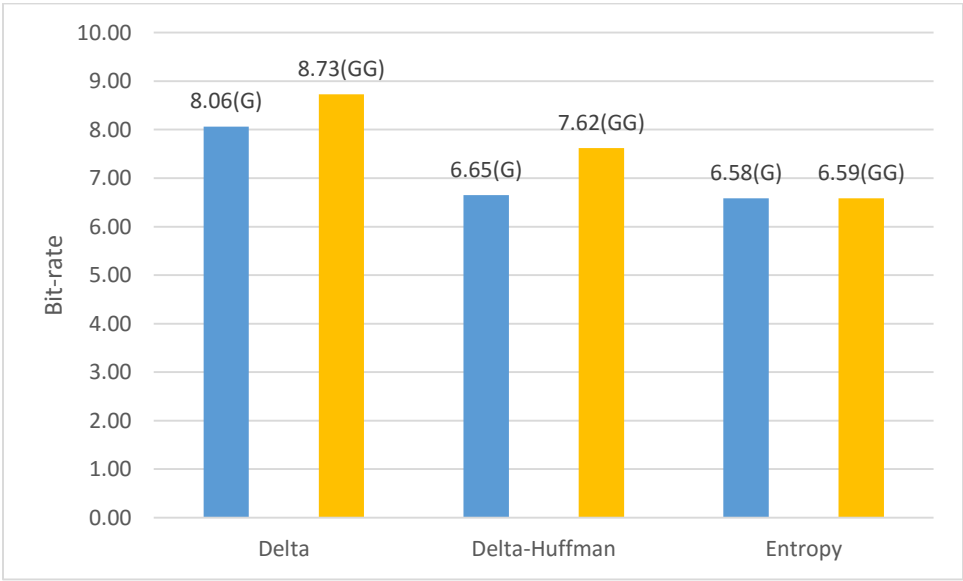


Figure 6.7: The bit-rate of State – gaps vs. gaps of gaps

Figure 6.8 provides the bit-rate of the compression algorithms Elias Delta code and δ-Huffman code for the gaps and gaps of gaps of inverted list – "Stephen".

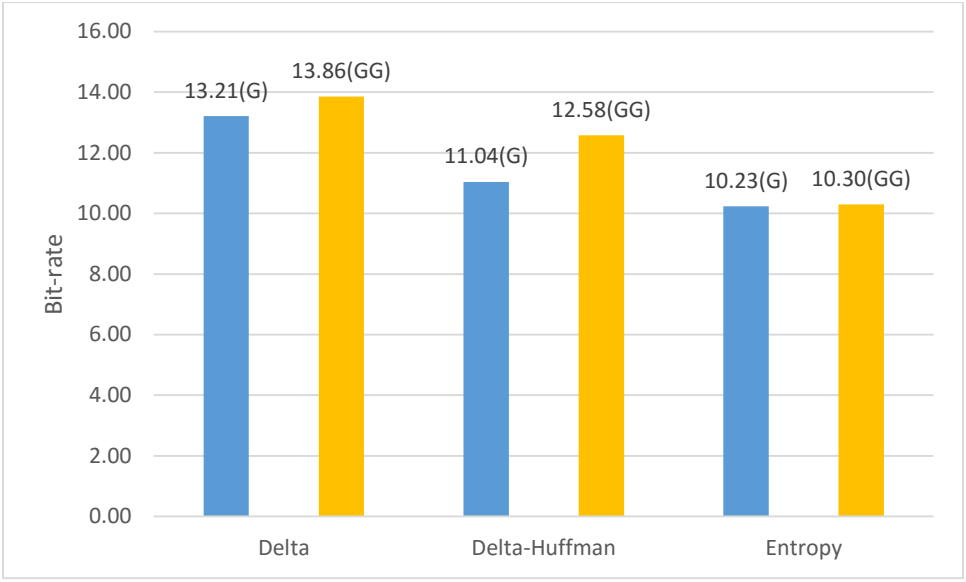


Figure 6.8: The bit-rate of Stephen gaps vs. gaps of gaps

Figure 6.9 provides the bit-rate of the compression algorithms Elias Delta code and δ-Huffman code for the gaps and gaps of gaps of inverted list – "Walker".

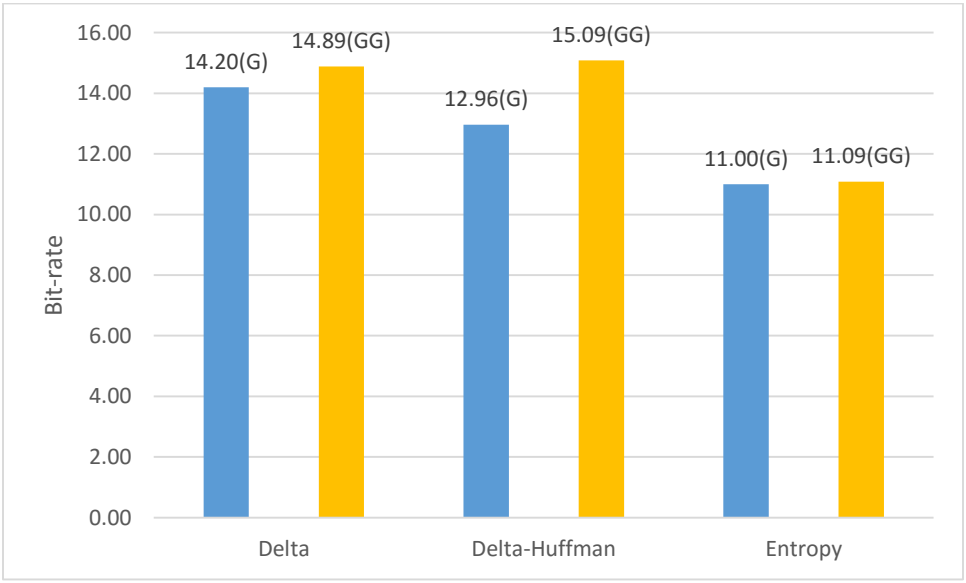


Figure 6.9: The bit-rate of Walker – gaps vs. gaps of gaps

Figure 6.10 provides the bit-rate of the compression algorithms Elias Delta code and δ-Huffman code for the gaps and gaps of gaps of inverted list – "War".

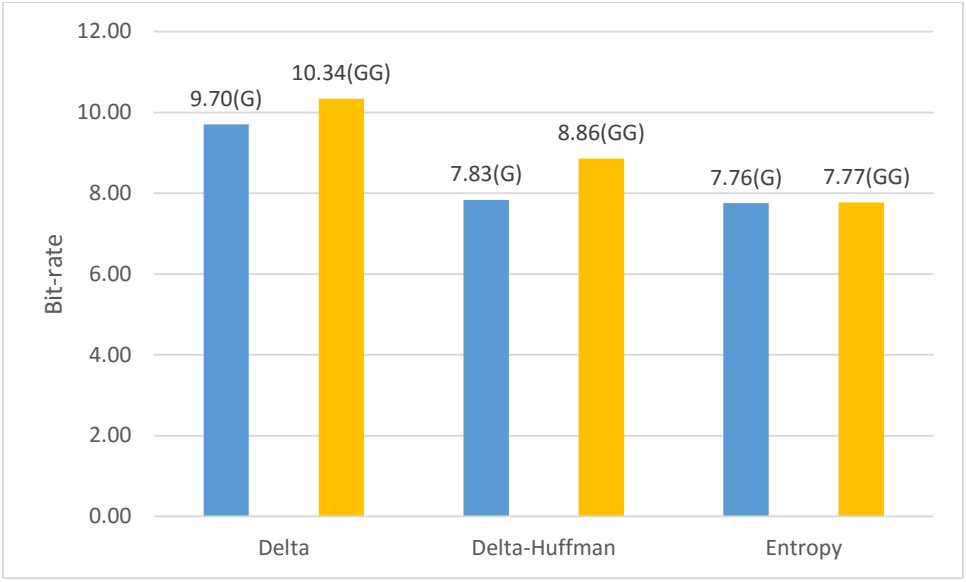


Figure 6.10: The bit-rate of War gaps vs. gaps of gaps

Figure 6.11 provides the bit-rate of the compression algorithms Elias Delta code and δ-Huffman code for the gaps and gaps of gaps of inverted list – "West".

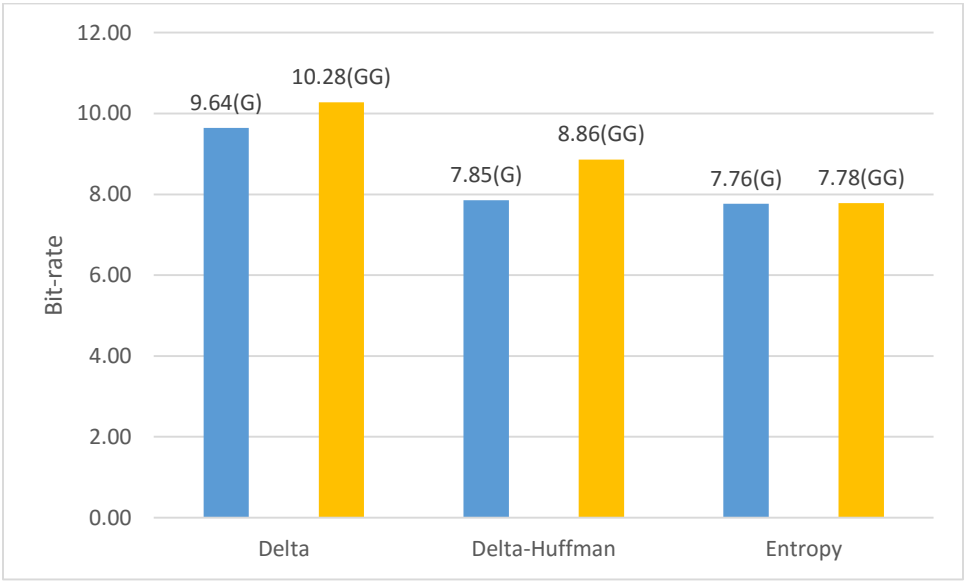


Figure 6.11: The bit-rate of West gaps vs. gaps of gaps

Figure 6.12 provides the bit-rate of the compression algorithms Elias Delta code and δ-Huffman code for the gaps and gaps of gaps of inverted list – "World".

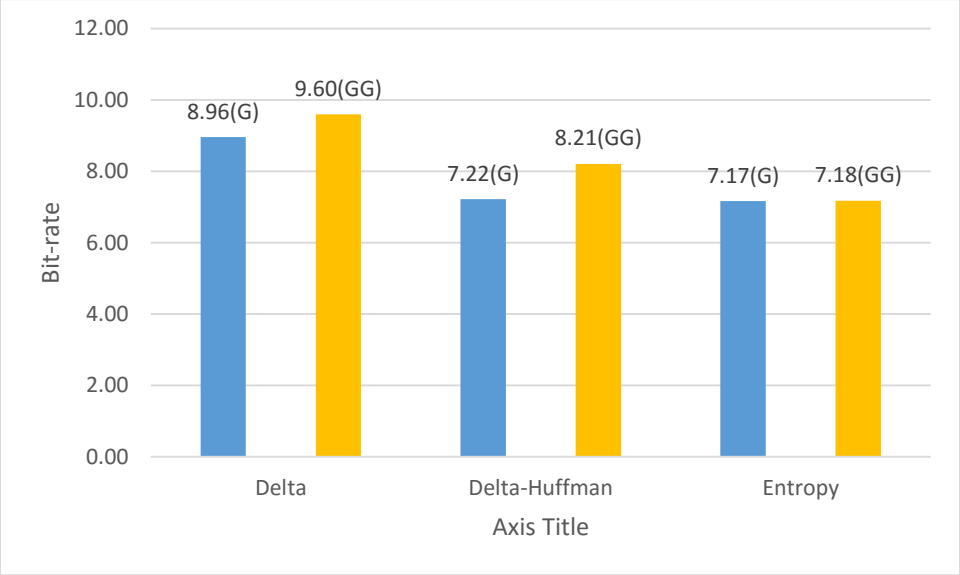


Figure 6.12: The bit-rate of World gaps vs. gaps of gaps

From figures 6.1, 6.2, 6.3, 6.4, 6.5, 6.6, 6.7, 6.8, 6.9, 6.9, 6.10, 6.11, and 6.12 we can observe that the better compression can be achieved from the gaps than gaps of gaps. We have used data-sets of inverted lists and experimented on gaps as well as gaps of gaps. We have applied the compression techniques such as Elias Delta code, δ-Huffman code and compared it to the entropy. In each case, gaps require fewer bits of compression compared to gaps of gaps. Comparing the compression techniques and the estimated entropy with respect to gaps and gaps of gaps, it can be observed that every compression technique for gaps requires fewer bits than gaps of gaps. By implementing gaps of gaps, the differences have produced wider range and this leads to without repetitions. Hence, gaps of gaps creates data-sets with larger integers, with no closer distribution of gaps and with very less repetitions. This explains that, the compression of gaps of gaps provides inferior bit-rate.

4.3.7 Experiment 7

In this section of experiments, we have compared and analyzed Elias Delta code and δ-Huffman code to the entropy on the gaps of sorted inverted lists vs. gaps of page-rank lists. For all the differences, we have implemented the sign and magnitude representation.

For all of the graphs depicted in this section, the left column bar (blue) is the compression techniques applied on the data-set of gaps of sorted inverted lists and the right column bar (yellow) is the compression techniques applied on the data-set of gaps of page-rank lists.

Below, we have provided the graphs of gaps of 4 sorted inverted lists vs. gaps of 4 page-rank lists. In all of the graphs, we have compared compression techniques Elias Delta code and δ-Huffman code to the entropy. In all of the following figures, "II" refers to gaps of inverted lists where as "PI" refers to page-rank lists.

Figure 7.1 provides the bit-rate of the compression algorithms Elias Delta code and δ-Huffman code for the gaps of inverted list and gaps of page-rank list – "Facebook".

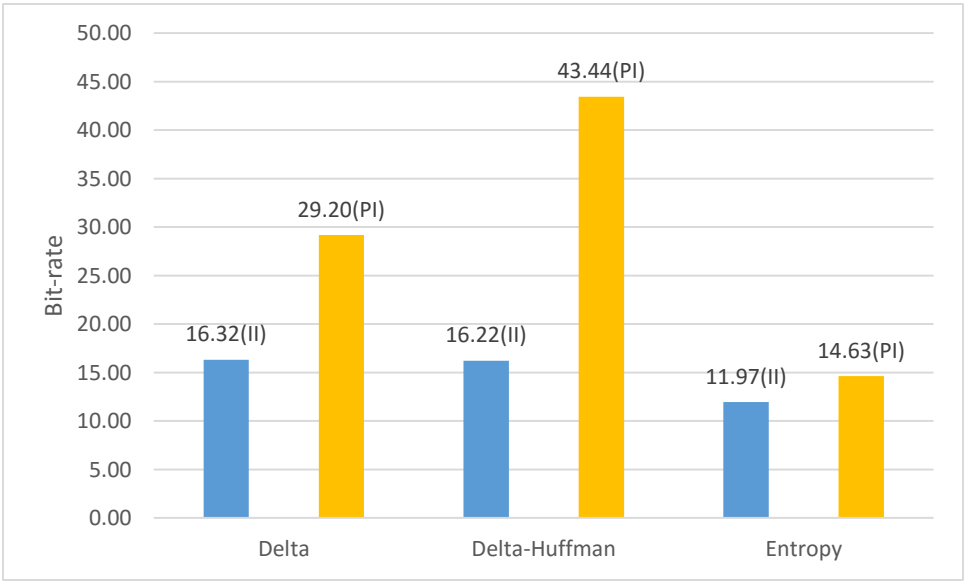


Figure 7.1: The bit-rate of Facebook II vs. PI

Figure 7.2 provides the bit-rate of the compression algorithms Elias Delta code and δ-Huffman code for the gaps of inverted list and gaps of page-rank list – "Grei".

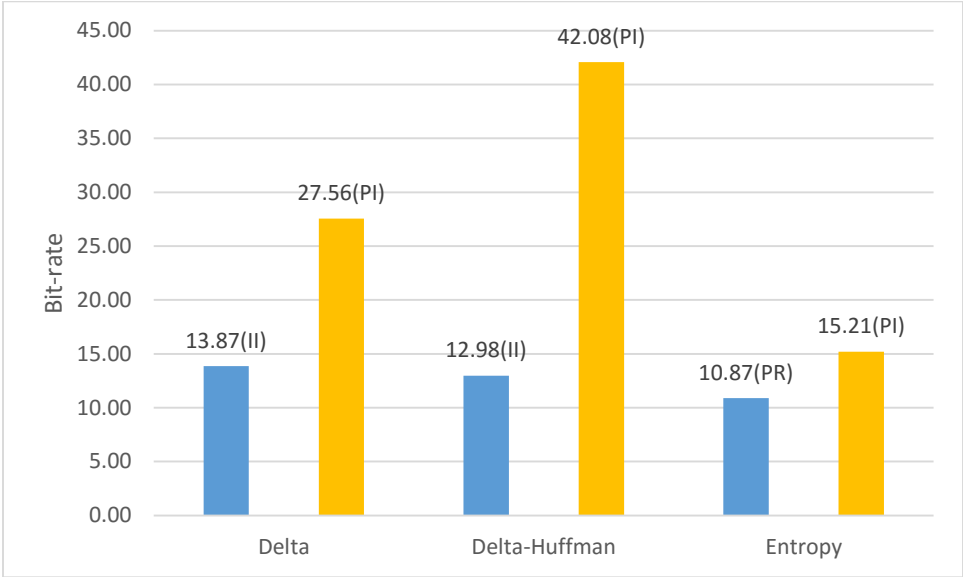


Figure 7.2: The bit-rate of Grei II vs. PI

Figure 7.3 provides the bit-rate of the compression algorithms Elias Delta code and δ-Huffman code for the gaps of inverted list and gaps of page-rank list – "Iraq".

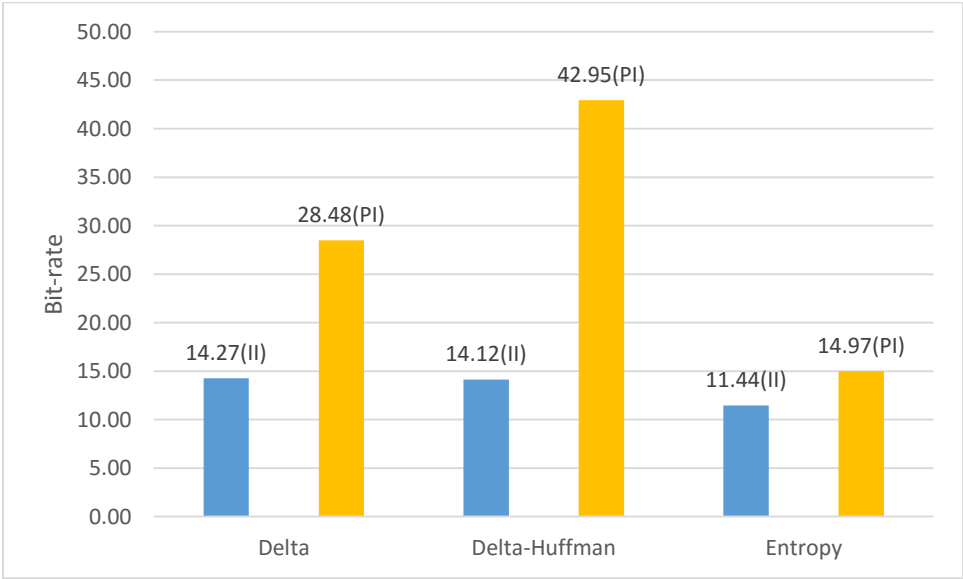


Figure 7.3: The bit-rate of Iraq II vs. PI

Figure 7.4 provides the bit-rate of the compression algorithms Elias Delta code and δ-Huffman code for the gaps of inverted list and gaps of page-rank list – "Obama".

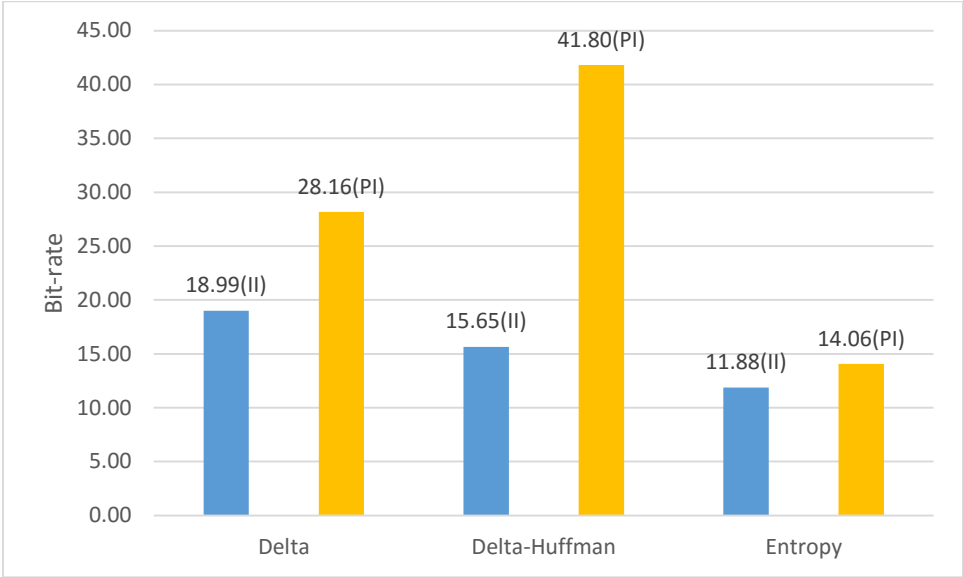


Figure 7.4: The bit-rate of Obama II vs. PI

From figures 7.1, 7.2, 7.3, and 7.4 we can observe that the compression techniques on the gaps of inverted lists require less bits than the gaps of page-rank lists. In all the cases, the entropy of gaps of gaps is higher than the entropy of gaps. By implementing gaps on page-rank lists, the data-sets range expands. This factor yields high bit-rate. Since the page-rank data is not ordered in monotonic way, compression of gaps of page-rank lists provides inferior results compared to the gaps of inverted lists.

4.3.8 Experiment 8

In this section, we have compared and analyzed the performance of compression algorithms "gzip," "bzip2," Elias Delta code, and δ-Huffman code on the data-sets of GPDF (0.5, 0.1, 0.01), PD, and on the gaps of sorted inverted lists.

Figure 8.1 provides the bit-rate of the compression algorithms gzip, bzip2, Elias Delta code, and δ-Huffman code for the GPDF (0.5, 0.1, 0.01), and PD.

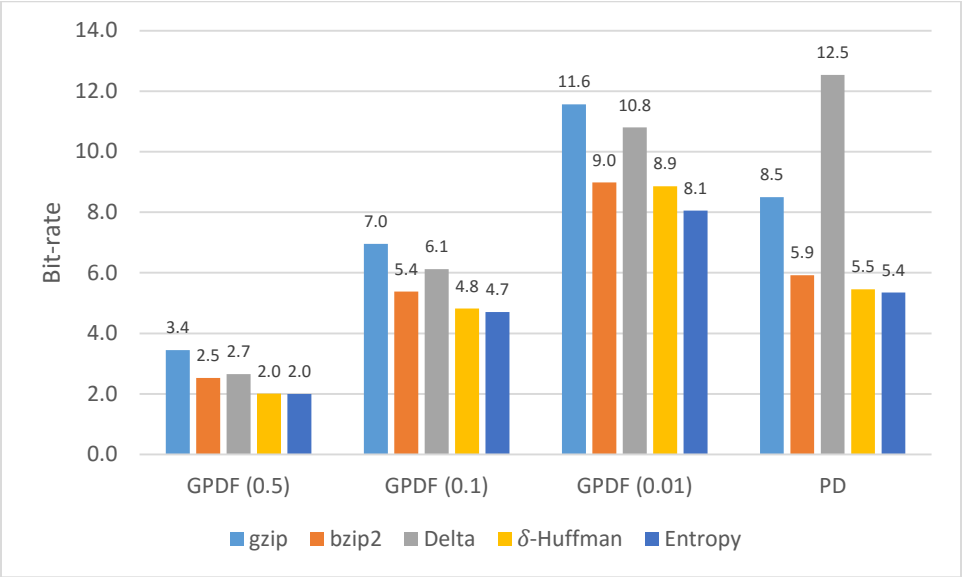


Figure 8.1: gzip vs. bzip2 vs. Delta vs. δ-Huffman – Distributions

Figure 8.2 provides the bit-rate of the compression algorithms gzip, bzip2, Elias Delta code, and δ-Huffman code for the gaps of inverted lists – "Bollywood," "Grei," and "Rousei."

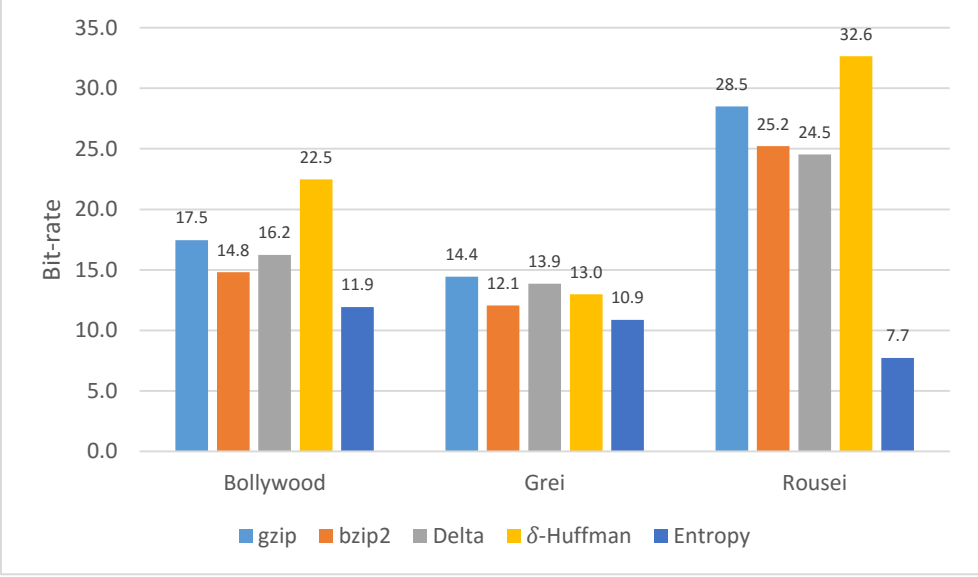


Figure 8.2: gzip vs. bzip2 vs. Delta vs. δ-Huffman – Inverted Lists (1/4)

Figure 8.3 provides the bit-rate of the compression algorithms gzip, bzip2, Elias Delta code, and δ-Huffman code for the gaps of inverted lists – "Walker," "Facebook," "Iraq," and "Obama."

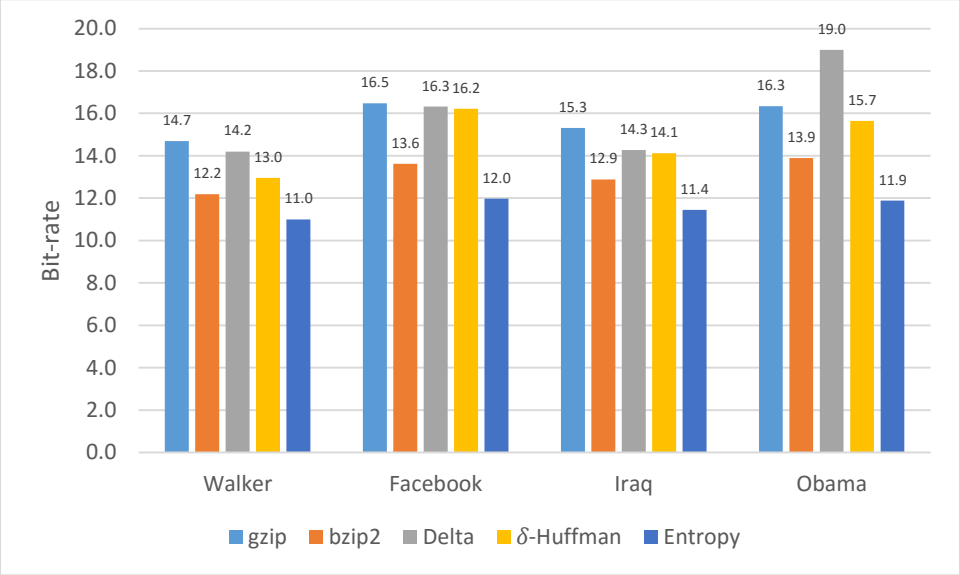


Figure 8.3: gzip vs. bzip2 vs. Delta vs. δ-Huffman – Inverted Lists (2/4)

Figure 8.4 provides the bit-rate of the compression algorithms gzip, bzip2, Elias Delta code, and δ-Huffman code for the gaps of inverted lists – "2015," "Film," "India," and "War."

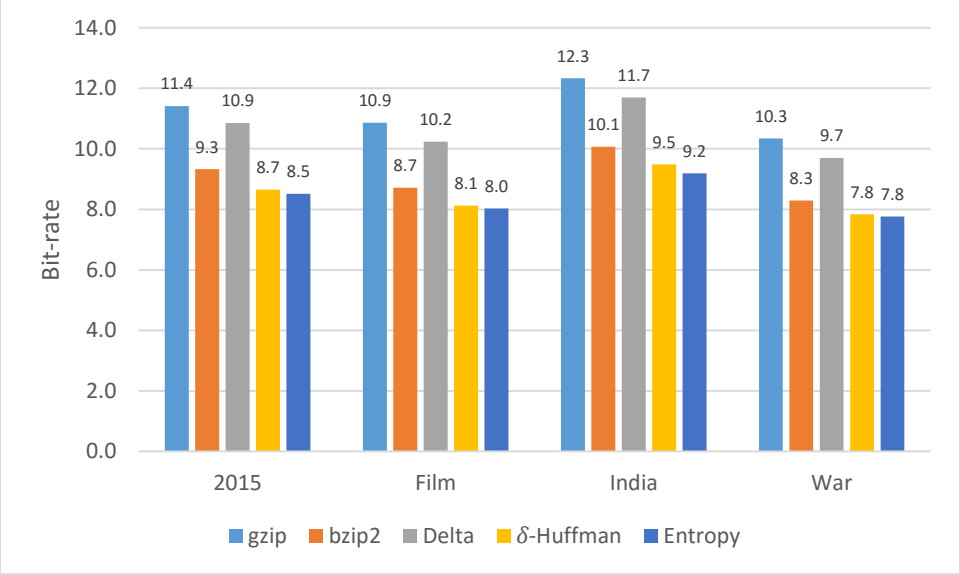


Figure 8.4: gzip vs. bzip2 vs. Delta vs. δ-Huffman – Inverted Lists (3/4)

Figure 8.5 provides the bit-rate of the compression algorithms gzip, bzip2, Elias Delta code, and δ-Huffman code for the gaps of inverted lists – "West," "World," "State," and "Stephen."

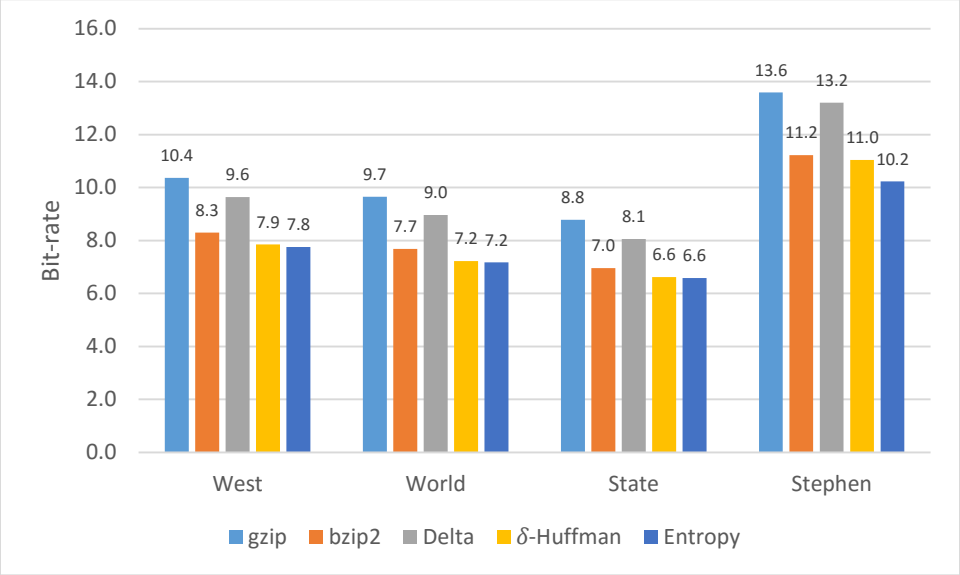


Figure 8.5: gzip vs. bzip2 vs. Delta vs. δ-Huffman – Inverted Lists (4/4)

From figure 8.1, we can deduce that in all of the distributions δ-Huffman code provides the lowest bit-rate among the compression techniques "gzip," "bzip2," and Elias Delta code and relatively close to the entropy. The reason is that δ-Huffman code provides comparatively low bit-rate when there are large data-sets containing small integers with numerous repetitions.

From figures 8.1, 8.2, 8.3, 8.4, and 8.5 we can observe that in every data-set δ-Huffman code is achieving low bit-rate than the compression technique "gzip". In the data-sets of gaps of inverted lists δ-Huffman code is achieving better bit-rate compared to the compression technique "bzip2" in "2015 (324,888 entries)," "Film (470,269 entries)," "India (193,810 entries)," "War (540,639 entries)," "West (521,325 entries)," "World (843,277 entries)." Whereas in the gaps of inverted lists of "Bollywood (10,606 entries)," "Grei (49,973)," "Rousei

(211 entries)," "Walker (50,874)," "Facebook (25,451)," "Iraq (32,488 entries)," and "Obama (17,153 entries)" "bzip2" is achieving the best bit-rate compared to the δ-Huffman code. From this, we can observe that for the large data-sets δ-Huffman code provides relatively better bit-rate compared to the other compression techniques. The reason is, the gaps produces closer intervals with numerous repetitions of integers. Where as in the case of smaller data-sets, there are very less repetitions of integers and δ-Huffman code provides inferior bit-rate.

4.3.9 Experiment 9

In this section, we have compared the performance of the compression algorithms Elias Delta code, δ-Huffman code, and Aging- Huffman (see section 4.1 Experimental setup) on the data-sets of GPDF (0.01) and on the gaps of sorted inverted lists of the terms "Obama," "Rousei," and "Walker."

Figure 9 provides the bit-rate of the compression algorithms Elias Delta code, δ-Huffman code, and Aging-Huffman for the data-set of GPDF (0.01) and data-sets of gaps of inverted lists – "Obama," "Rousei," and "Walker."

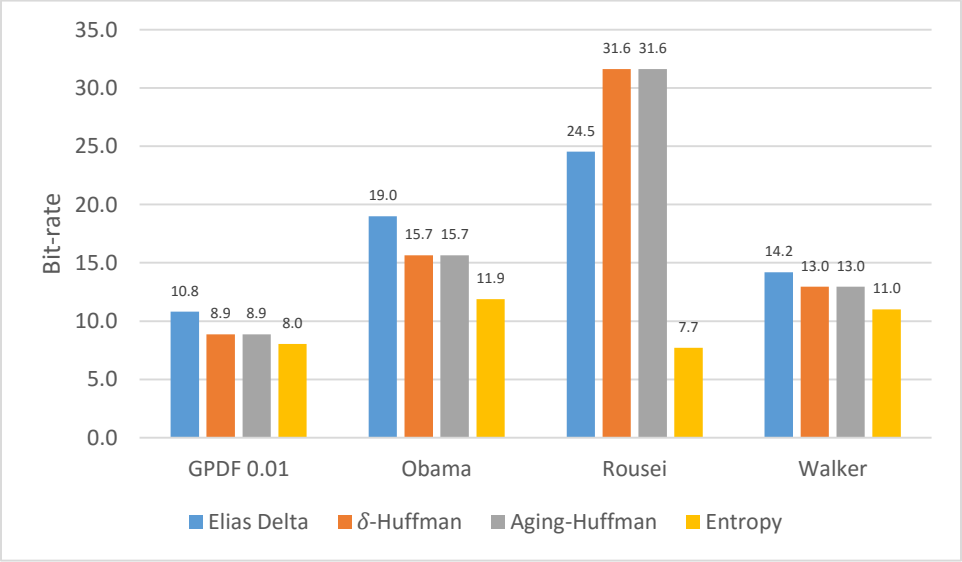


Figure 9: Elias Delta vs. δ-Huffman vs. Aging-Huffman – Inverted Lists

From the above figure we can observe that on the data-sets examined the performance of δ-Huffman code is equal to Aging-Huffman. This can be explained via the observation that in these data-sets there are frequently occurring numerous repetitions and the count is not decremented. In future work we plan to further examine the Aging-Huffman algorithm with several aging parameters.

CHAPTER 5

RESULTS EVALUATION

The following observations were made based on the results of the experiments performed in this thesis research.

- Golomb code with high parameters (e.g., 16) provides low bit-rate for linearly increasing integers. For small data-sets with small input integers, Golomb code with low value parameter (e.g., 2) is a good fit. Additionally, in order to get a better bit-rate for the Golomb code, knowledge on the data-sets should be obtained prior to the actual compression. This might necessitate a two-pass or offline implementation.
- Comma code is a good option for compressing large data-sets which contain no repetitions of input integers. For small data-sets with small input integers, Comma code exhibits low compression ratio.
- Among the Elias codes, for large data-sets with medium to large input integers and few repetitions, Elias Delta code provides the best bit-rate. This also holds, for large data-sets with small input integers and high level of repetitions. Elias Gamma code provides the best bit-rate for small data-sets with small input integers.
- Fibonacci code provides low bit-rate for medium to large data-sets with medium to large input integers and with few repetitions.
- δ-Huffman code provides the best bit-rate, which is almost equal to the entropy, in the data-sets of GPDF and PD, gaps of GPDF and PD, as well as gaps of gaps of GPDF and PD. The same holds for the gaps of sorted inverted lists, gaps of gaps of sorted inverted lists. δ-Huffman code provides better bit-rate than "gzip" in every experiment performed.

- Compared to "bzip2," δ-Huffman code produces a better bit-rate in large data-sets with numerous repetitions. In the small data-sets with no repetitions, "bzip2" provides the best bit-rate, whereas δ-Huffman code provides bit-rate that is smaller than "bzip2" but higher than "gzip". Note, however, that "bzip2" and "gzip" are two-pass algorithms while the δ-Huffman is a single pass procedure.
- Compression of gaps of large data-sets provides better compression than compression of gaps of small data-sets, because the gaps are in clustered in closer range.
- For the signed gaps, e.g., those obtained from the unsorted data, the sign and magnitude representation provides lower bit-rate than the odd-even mapping. With the sign and magnitude representation, the δ-Huffman compression of the gaps provides bit-rate that is almost equal to the entropy. This is explained by the fact that the δ-Huffman is highly efficient with small integers but the odd-even mapping produces relatively large integers.
- The δ-Huffman code provides high bit-rate for page-rank lists because the page-rank lists have very large input integers containing none or very few repetitions. In this case, Comma code provides relatively low bit-rate.
- The δ-Huffman code is the lowest performer in small data-sets with few or no repetitions.
- The bit-rate of the δ-Huffman code required to compress gaps of large data-sets of sorted inverted lists is almost equal to the bit-rate of the δ-Huffman code require to compress data-set of GPDF (0.01).
- Many of the algorithms discussed above require apriori knowledge concerning the data-set and the best parameters for compressing it. In contrast the δ-Huffman is a single pass algorithm that does not require prior knowledge about the data.

CHAPTER 6

CONCLUSION AND FUTURE WORK

This thesis presents an approach to increase the achievable compression ratios in integer compression algorithms. Compression ratio is our main emphasis and we are less concerned with the computation complexity. We have devised a new integer coding algorithm, δ-Huffman coding, by exploiting the properties of Elias Delta code and Huffman code and combining the two methods. To the best of our knowledge, this is the first work that combines the unbounded integer compression methods proposed by Elias with Huffman coding and comparatively evaluates their performance. δ-Huffman code can be used effectively for IR applications via lossless data compression. To evaluate the utility of δ-Huffman in IR applications, we have applied the compression algorithms on numerous data-sets drawn from GPDF, PD, sorted inverted lists, and page-rank lists obtained from Wikipedia. Our experimental results demonstrate that the bit-rate of δ-Huffman code is very close to the estimated entropy and it is the best among other integer compression techniques such as Comma code, Elias Gamma code, Elias Delta code, Elias Omega code, Fibonacci code, and Golomb code. δ-Huffman code provides the best bit-rate compared to the compression technique "gzip" and better than "bzip2" in most of the cases, when there are large data-sets with numerous repetitions.

We plan to further explore the algorithms detailed in this thesis and evaluate other features of their performance including asymptotic optimality as well as cost/effectiveness in terms of compression ratio, throughput, latency, energy consumption, and implementation cost. Furthermore, we plan to expand the set of experiments to include additional methods. Moreover, we plan to further examine the Aging-Huffman algorithm with several aging

parameters. Finally, we plan to explore practical (yet not necessarily optimal) algorithms as well as dynamic unbounded integer compression when the probability of occurrence of integers is known.

LITERATURE CITED

1. Prabhakar Raghavan, Hinrich Schutze Cambridge, *An Introduction to Information Retrieval*, Cambridge University Press.

2. Shannon Claude E, "A Mathematical Theory of Communication", *Bell System Technical Journal*, PP. 27:379-423, 623-656, 1948.

3. Huffman David, "A Method for The Construction of Minimum Redundancy Codes" Proceedings of the IRE, vol. 40(9) pp. 1098-1101, 1952.

4. V. Glory, S. Domnic, "Inverted Index Compression Using Extended Golomb Code" in Advances in Engineering, Science and Management (ICAESM), International Conference, IEEE, PP. 20-25, 2012.

5. S. David, *Data Compression*, 2nd ed., Springer, New York, pp. 41-113, 2000.

6. P. Elias, "Universal Code Word Sets and Representations of The Integers". *IEEE Transactions on the Information Theory*, vol. IT-21(2), pp.194-203, March 1975.

7. S. Golomb, "Run-length encodings". *IEEE Transactions on Information Theory,* vol. 12(3), pp. 339-401, 1966.

8. D. Tamir, "Elias Omega Coding", *ACM ICPC Contest* Asia-Phuket 2009/2010.

9. Vo Ngoc Anh, Alistair Moffat, "Index Compression using Fixed Binary Codewords" in ADC '04 *Proceedings of the 15th Australasian database conference* - Volume 27, Australian Computer Society, Inc. Darlinghurst, Australia, PP.61-67, 2004.

10. Falk. Scholer, Hugh E. Williams, "Compression of Inverted Indexes for Fast Query Evaluation" in SIGIR '02 *Proceedings of the 25th annual international ACM SIGIR conference on Research and development in information retrieval*, PP.222-229, 2002.

10. Falk. Scholer, Hugh E. Williams, "Compression of Inverted Indexes for Fast Query Evaluation" in SIGIR '02 *Proceedings of the 25th annual international ACM SIGIR conference on Research and development in information retrieval*, PP.222-229, 2002.

11. A. Yang, H. Mukka, F. Hesaaraki, and M. Burtscher. "MPC: A Massively Parallel Compression Algorithm for Scientific Data". *IEEE Cluster Conference*. September 2015.

12. M. Burtscher and P. Ratanaworabhan. "gFPC: A Self-Tuning Compression Algorithm. *2010 Data Compression Conference"*, pp. 396-405. March 2010.

13. M. Burtscher and P. Ratanaworabhan. "pFPC: A Parallel Compressor for Floating-Point Data". *2009 Data Compression Conference*, pp. 43-52. March 2009.

14. M. Burtscher and P. Ratanaworabhan. "FPC: A High-Speed Compressor for Double-Precision Floating-Point Data". *IEEE Transactions on Computers*, Vol. 58, No. 1, pp. 18-31. January 2009.

15. M. Burtscher and P. Ratanaworabhan. "High Throughput Compression of Double-Precision Floating-Point Data". *2007 Data Compression Conference*, pp. 293-302. March 2007.

16. John Jenkins and Isha Arkatkar. "ALACRITY: Analytics-Driven Lossless Data Compression for Rapid In-Stu Indexing, Storing, and Querying", pp. 95-114. *2013*.

17. Daniel Lemire and Leonid Boytsov. "Decoding Billions of Integers Per Second Through Vectorization". September *2012*.

18. Marcin Zukowski, Sandor Heman, "Super-Scalar RAM-CPU Cache Compression: PFORDELTA" in SIGIR '06 *Proceedings of the 22nd International Conference on Data Engineering*, PP.59, 2002.

19. Sayood Khalid. *Introduction to Data Compression.* San Francisco, CA: Morgan Kauffmann, 2012.

20. Saloman, David. *Data Compression, The Complete Reference.* December 19, 2006

21. *The gzip home page*. 2006. http://www.gzip.org/.

22. *The bzip2 home page*. 2006. http://www.bzip2.org/.