# PERFORMANCE OF DELTA HUFFMAN COMPRESSION VARIANTS

by

Bryant Aaron, 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
December 2020

Committee Members:

Dan Tamir, Chair

Jelena Tesic

Vangelis Metsis

## FAIR USE AND AUTHOR’S PERMISSION STATEMENT

### Fair Use

### Duplication Permission

## DEDICATION

Dedicated to my parents.

# ACKNOWLEDGEMENTS

Acknowledgements to Dr. Tamir, the committee members on my thesis, and all of the Texas State faculty and staff for helping me in this endeavor.

# TABLE OF CONTENTS

**Page**

# LIST OF TABLES

**Table** **Page**

## LIST OF FIGURES

**Figure** **Page**

# ABSTRACT

This thesis examines an algorithm that is a combination of the Elias Delta and Huffman algorithms known as $\delta$-Huffman. Exploiting the fact that Elias Delta codes are uniquely decodable and tend to allocate fewer bits to small integers, while Huffman coding is a lossless compression method that provides compressed symbols with a bit rate that is relatively close to the entropy of the source.

The thesis “Dynamic Unbounded Integer Compression” by Naga Sai Gowtham Mulpuri explored achievable compression ratios in the context of $\delta$-Huffman encoding and used the frequency of inputs as the estimation for the probability distribution of the $\delta$-Huffman algorithm on synthetic data and real-world posting list data.

Our work focuses on different assumptions about the way to estimate the probability function for the $\delta$-Huffman algorithm. Assumptions such as using a Geometric Probability Mass Function in the estimation of the probability distribution of the $\delta$-Huffman algorithm are made and methods such as slice and reset with the $\delta$-Huffman algorithm are introduced.

We examine the practical and theoretical bounds on the compression capabilities of integer compression methods. Additionally, we devise methods for efficient compression of integers regardless of their specific probability distribution function. The focus of our work has been on the $\delta$-Huffman algorithm’s compression, rather than on its computation complexity.

Twelve experiments with twelve variants of the dynamic $\delta$-Huffman algorithm are performed. The experiments use one or more of three source data sets: synthetic data from several Geometric Probability Mass Functions (GPMF) and Poisson, real-world data from sorted inverted index gaps from Wikipedia, and a benchmark that attempts to represent realistic workload data from Silesia. The entropy estimates of these data sets and the average bit rate of the $\delta$-Huffman algorithms offer a way to construct a comparative evolution of the performance of these algorithms.

From these experiments, the best variant out of the twelve $\delta$-Huffman algorithms for synthetic data, real-world data, and benchmark data are determined.

Finally, given the results of the experiments, assuming GPMF without knowledge concerning the exact value of parameters, the best variant out of the twelve $\delta$-Huffman algorithms is identified

# 1. INTRODUCTION

This thesis examines different assumptions about the estimation of the probability function for the $\delta$-Huffman algorithm ([15]) and looks at experimental results for finite or infinite data sets to evaluate the theoretical and practical utility of the $\delta$-Huffman algorithm.

Work by Mulpuri shows how the new $\delta$-Huffman algorithm can be effective for Information Retrieval (IR) applications via its lossless data compression capability [15]. Comparatively, there are fewer experiments with the $\delta$-Huffman algorithm in [15] than in this work. Experiments performed in [15] make only one assumption for the estimation of the probability function of the $\delta$-Huffman algorithm. Finally, [15] only performs experiments with the $\delta$-Huffman algorithm on synthetic data and real-world posting list data. However, the experimental results in [15] does show the bit rate values of the $\delta$-Huffman algorithm to be very close to the entropy of the data source and that 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. Experimental results in [15] also show that the $\delta$-Huffman algorithm provides the best bit rate value versus the “gzip” and “bzip2” compression algorithms in most of the cases when there are large data sets.

The $\delta$-Huffman algorithm is worth further exploration, and this paper further explores the $\delta$-Huffman algorithm through manipulation of the estimation of the probability function to examine the practical compression capabilities of integer compression methods and the theoretical bounds (i.e., universality and asymptotical optimality) on the compression capabilities of integer compression methods devised by

Tamir et al. and to devise methods for efficient compression of integers regardless of their specific probability distribution function [18].

This thesis empirically explores twelve different approaches for the estimation of the probability function for the $\delta$-Huffman algorithm to identify the theoretical and practical capabilities of dynamic integer compression methods with synthetic and realistic data. The experiments analyze and compare the performance of static compression and dynamic compression techniques with variants of $\delta$-Huffman. In addition, this paper analytically evaluates the theoretical performance of these variants.

The hypothesis of this work is that these several variants of the $\delta$-Huffman algorithms can be highly efficient for practical applications; they possess the property of asymptotically optimality and can be augmented with efficient update procedures.

To the best of the writer's knowledge, this is the first attempt to primarily address the estimation of the probability function for the $\delta$-Huffman algorithm and to develop update algorithms and establish practical and theoretical performance measures.

The rest of this thesis is as follows: Chapter 2 provides background information; Chapter 3 contains the literature review; Chapter 4 outlines the experiments; Chapter 5 examines the experimental results; Chapter 6 evaluates the results, and Chapter 7 concludes and considers future work.

# 2. BACKGROUND

This chapter explains the relevant data and integer compression terms in this paper.

## 2.1 Data Compression

Data compression reduces the number of bits required to store or transmit data. This section covers common data compression terms.

**2.1.1 Bit Rate and Compression Ratio.** The bit rate ($BR$) is the average number of bits needed to transmit input symbols from a data source.

$BR = \frac{total\ number\ of\ bits\ sent}{number\ of\ input\ symbols}.$

The compression ratio ($CR$) is the ratio of the size of the input stream to the output stream. $CR = \frac{size\ of\ input\ stream}{size\ of\ output\ stream}$ [7].

**2.1.2 Entropy.** The entropy is the theoretical lower bound on the communication bit rate over a noiseless communication channel. The entropy ($H$) of a source $X$ with alphabet $A_x = \{a_1 \ldots a_n\}$ and probabilities ($p_1 \ldots p_n$) is given by $H(X) = -\sum_{i=1}^{n} p_i \log_2(p_i)$ [1].

**2.1.3 Uniquely Decodable.** A uniquely decodable code is a code that can be decoded in one, and only one, way [1].

**2.1.4 Prefix Code.** A prefix code is code that has no code-word that is a prefix to another code-word [1].

**2.1.5 Universal Code.** A universal code is a code that compresses messages, regardless of the probability distribution function of these messages, to code-words whose average length is bounded by $C_1H_1 + C_2$ and $C_1$ and $C_2$ are constants greater than or equal to 1. It is asymptotically optimal if the constant value $C_1 = 1$.

**2.1.6 Fixed-Length Code.** A fixed-length code (FLC) is a code that uses the same number of bits or bytes to represent each symbol from a data source. Fixed-length codes are also known as block codes [2]. The benefit of FLC is the ease of use (encoding and decoding). The issue, however, is that FLC addresses the worst case scenario and does not enable compression via assignment of a smaller length code to the most probable symbols.

## 2.2 Types of Compression

**2.2.1 Variable-Length Code.** A variable-length code (VLC) is a code that can vary in length [2]. Their advantage is that they can allocate the smallest possible code to the most common symbols, which on average enables better bit rates and compression ratios.

**2.2.2 Lossless Compression.** A lossless compression is a compression technique that involves no loss of information. If data have been losslessly compressed, the original data can be recovered exactly from the compressed data. Lossless compression is generally used for applications that cannot tolerate any difference between the original and reconstructed data [1].

**2.2.3 Lossy Compression.** A lossy compression is a compression technique that involves loss of information. Data that have been compressed with lossy compression generally cannot be recovered or reconstructed exactly. In return for this distortion in the reconstruction, we can generally obtain much higher compression ratios than is possible with lossless compression [1].

**2.2.4 Bit-level Compression and Byte-level Compression.** The bit-level compression generates a variable number of bits for each input symbol while the byte-level compression encodes symbols' bits and/or bytes into a variable number of bytes.

## 2.3 Compression Algorithms

This section discusses the different compression methods for the algorithms in this thesis and explains the details of the algorithms.

**2.3.1 Elias Gamma Code.** The Elias Gamma code can encode positive unbounded integers. Given a non-negative integer $n$, the Elias Gamma representation of $n$ involves the binary equivalent of $n$, $B(n)$, and the length of the bit string that represents the binary equivalent of $n$, that is $|B(n)|$. The final Elias Gamma representation is $(|B(n)| - 1)$ number of zeroes than the binary representation of the given number [3].

For example, to find the Elias Gamma code for $n = 4$, first find the binary equivalent of 4, which is $B(4) = 100$. The length of the bit string that represents the binary equivalent of 4, that is $|B(4)| = 3$. The final Elias Gamma representation is $|B(4)| - 1$ number of zeroes than the binary representation of the given number $n = 4$, which is 00100 and the first two zeroes are from the result of $(|B(4)| - 1)$.

Elias Gamma code provides high compression when the data set has mainly integers of smaller values versus integers of larger values. Elias Gamma code is a universal code.

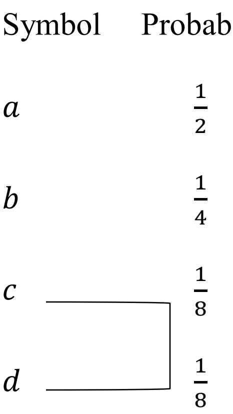
**2.3.2 Elias Delta Code.** The Elias Delta code further encodes the Elias Gamma code for a given non-negative number $n$. In general, the Elias Delta code replaces the first field of zeroes of the Elias Gamma code with the Elias Gamma code of that field [3]. Elias Delta code is relatively efficient and is asymptotically optimal.

**2.3.3 Elias Omega Code and Huffman Code.** The Elias Omega code applies recursion that employs Elias Gamma code on the $|B(n)|$ element. The Huffman code uses code-words that are not a prefix to another code-word. Huffman code is used when the probability values of the data in the source sequence are known. The Huffman code can be represented as a binary tree in which the external nodes or leaves correspond to the symbols [1].

Example: A source contains the symbols $\{a, b, c, d\}$ and the known probability values for the source data input of ⟨adcabacb⟩ are $[p(a) = \frac{1}{2}, p(b) = \frac{1}{4},\ p(c) = \frac{1}{8}, p(d) = \frac{1}{8}]$.

The Huffman code procedure follows:

1. Generate a list for the symbols with the probability values decreasing in value.



| Symbol | Probability Value |
|---|---|
| $a$ | $\frac{1}{2}$ |
| $b$ | $\frac{1}{4}$ |
| $c$ | $\frac{1}{8}$ |
| $d$ | $\frac{1}{8}$ |

**Figure 2.1: Probability value step 1**

2. Combine the two symbols that have the two smallest probability values. If the new symbol has the same probability value as another symbol in the list, place the new symbol at the head of the order in the list for that probability value.

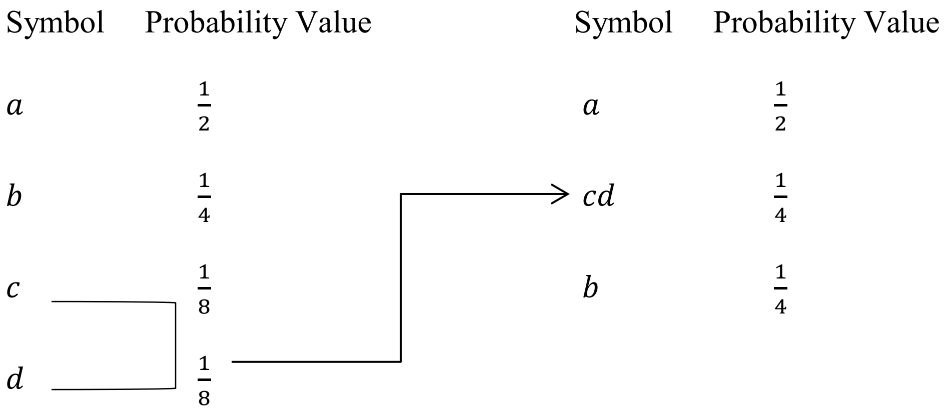


**Figure 2.2: Probability value step 2**

3. Continue to combine symbols until only one symbol remains in the list.

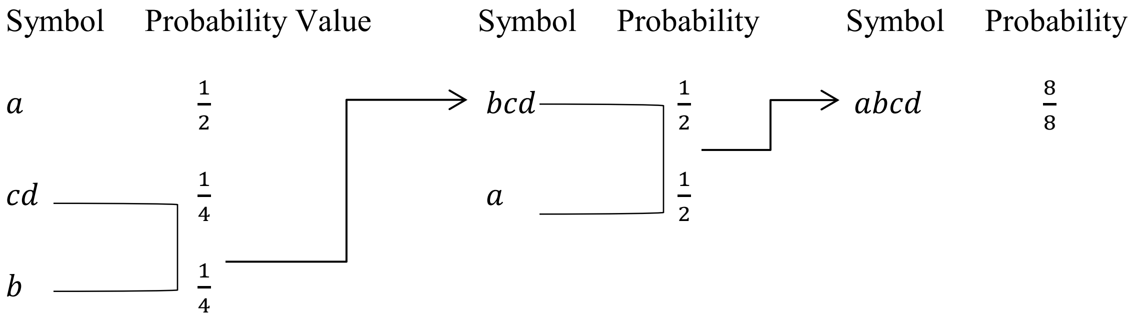


**Figure 2.3: Probability value step 3**

4. Work in reverse order to generate the Huffman tree. The number of bits in the Huffman code of a symbol is the depth of that symbol in the Huffman tree and its code is a description of the path from the root of the Huffman tree to the symbol (e.g., use 0 for a left branch and 1 for a right branch).

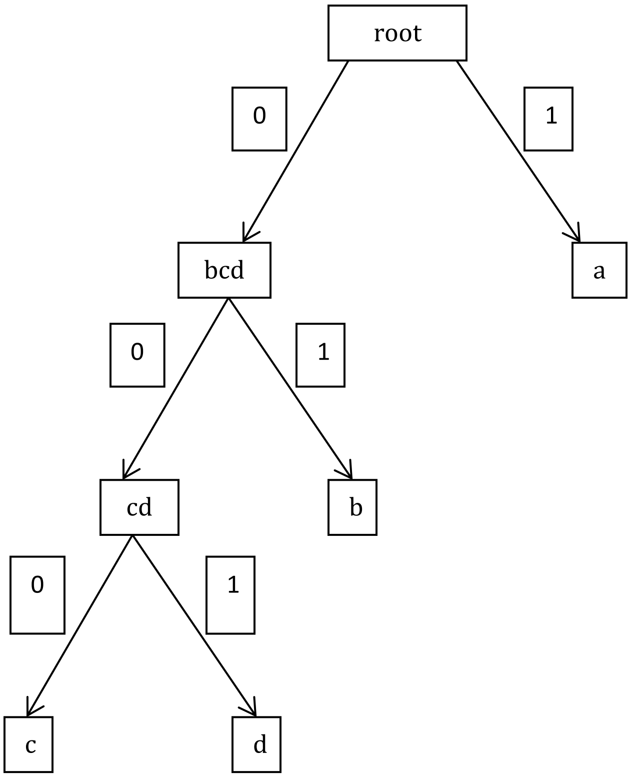


**Figure 2.4: Huffman Tree**

**Table 2.1: Huffman Codes**

| Symbol | Huffman Code |
|---|---|
| *a* | *1* |
| *d* | *001* |
| *c* | *000* |
| *a* | *1* |
| *b* | *01* |
| *a* | *1* |
| *c* | *000* |
| *b* | *01* |

The final Huffman code for the source input ⟨adcabacb⟩ is “1001000101100001”.

**2.3.4 Dynamic Huffman Code.** The dynamic Huffman code is used when the probability values of the data in the source sequence are unknown. The code process of the dynamic Huffman code estimates these probability values via accumulated weights as the data arrives at the encoder. The initial Huffman tree for the dynamic Huffman code has only one special node to represent the list of all data which are “not yet transmitted” by the dynamic Huffman code known as the NYT node. Generally, the NYT node has a permanent weight value of 0 in the Huffman tree.

**2.3.5 δ-Huffman.** The $\delta$-Huffman is a combination of Elias Delta and Huffman compression algorithms. Elias Delta uniquely encodes unknown data (e.g., unbounded integers) and Huffman assigns a code value to the resultant Elias Delta data. The combination of the two allows for a one pass compression of unbounded data.

$\delta$-Huffman works as follows. The NYT initially contains all data. When the first data element $n$ arrives at the encoder, $n$ is removed from the NYT and placed into an ‘already transmitted’ list (AT) with a weight value of 1. The encoder transmits the Huffman code of the NYT along with the Elias Delta code of $n$ to the decoder [3]. Then, with the new AT value, the encoder and decoder update their Huffman trees.

When an integer $n$ that is already in the AT list arrives, its Huffman code is sent to the decoder. Next, the weight value of $n$ in the AT list increases by 1. Finally, with the new AT value, the encoder and decoder update their Huffman trees. Although enforcement of the sibling property can provide an efficient way to update, it is not strictly enforced in this work as we are not concerned about the computational complexity of the update process.

**2.3.6 δ-Huffman Probability Formulas.** The $\delta$-Huffman probability formulas are used by the $\delta$-Huffman algorithms in this thesis to update their Huffman trees. This includes $(p_n) = \frac{frequency\ count\ of\ n}{sum\ of\ all\ frequency\ counts}$, and,

$$(p_n)$$

The dynamic probability formula one (P1) $= (1 - \alpha)^{(n-1)} * \alpha$ , and,

$i$ is the current iteration of the $\delta$-Huffman algorithm experiment,

$n$ is data from an input source that reaches the encoder,

$\sum_0^i n$ is the sum of all data $n$ in a segment of data,

And is given by $= \frac{i}{\sum_0^i n}$.

(P1)

The dynamic probability formula two (P2) $= \frac{n_c}{N}$ , and,

$n$ is data from an input source that reaches the encoder,

$c$ is the total count of a data $n$,

$n_c$ is the total count of the $n^{th}$ data,

And $N$ is given by the sum of all data $n$.

(P2)

## 2.4 Inverted Index

**2.4.1 Inverted Index.** The inverted index is the first major concept in information retrieval (IR) [4]. An index always maps back from terms to the parts of a document where they occur. The basic idea of an inverted index is to keep a dictionary of terms, then for each term, there is a list that records which documents the term occurs in. Each item in the list, which records that a term appeared in a document, is conventionally called a posting.

For example:

| [ Dictionary ] | | [ Postings ] |
|---|---|---|
| [ grei ] | → | [ 621 ][ 628 ][ 632 ][ 717 ][ 802 ][ 803 ][ 958 ] |
| [ rousei ] | → | [ 10846 ][ 15601 ][ 32579 ][ 34760 ][ 43564 ][ 128800 ] |
| [ state ] | → | [ 12 ][ 25 ][ 303 ][ 305 ][ 307 ][ 308][ 309][ 324 ][ 330 ][ 332 ] |

**2.4.2 Gap Construction.** The gap construction is the process of attaining the differential change between the consecutive document IDs [5]. Since the IDs are unique and are sorted by the web-crawler, we cannot get a value of 0. Furthermore, this process adds redundancy to a given posting list, and, generally, the gaps between indexes are smaller on average than the index themselves. Hence, the distribution of gaps "prefers" small integers.

For example, given the first 10 items of the posting list for the Wikipedia search term "$state$" is: $\langle \text{"state"}\ ;\ 12, 25, 303, 305, 307, 308, 309, 324, 330, 332\rangle$, the result after gap construction is: $\langle \text{"state"}\ ;\ 12, 13, 278, 2, 2, 1, 1, 15, 6, 2\ \rangle$. Notice that the gap index has multiples of the same number and a smaller average index value.

## 2.5 Distributions

**2.5.1 Geometric Probability Mass Function (GPMF).** The geometric probability mass function is the probability distribution of the number of failures in a random event before the first success; support is on the set $\{0, 1, 2, 3\ \cdots\}$. It is the probability that the first occurrence of success, in a set of binary independent trials, occurs after $X$ trials, each with a success probability $p$. It is given by Probability $(X = k) = (1-p)^{k-1}p, for\ k\ = 1,2,3, \ldots$

**2.5.2 Poisson Mass Function (PMF).** The Poisson mass function is a discrete probability distribution function that expresses the probability of a given number of events to occur 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 $P(k\ events\ in\ interval) = \frac{\lambda^k e^{-\lambda}}{k!}$ and λ is the average number of events per interval.

## 2.6 Sibling property

**2.6.1 Sibling property.** The sibling property is when a binary tree with $p$ leaves of nonnegative weight is a Huffman tree and

(1) The $p$ leaves have nonnegative weights $w1, \ldots, wp$, and the weight of each internal node is the sum of the weights of its children; and

(2) The nodes can be numbered in non-decreasing order by weight, so that nodes $2j - 1$ and $2j$ are siblings, for $1 \leq j \leq p - 1$, and their common parent node is higher in the numbering [6].

## 2.7 Types of δ-Huffman Algorithms

This section discusses the different assumptions made by the $\delta$-Huffman variants for the probability estimation for the $\delta$-Huffman algorithm. Except where discussed, the $\delta$-Huffman variants assume the encoder and decoder dynamically update the probability value as the last step in the algorithm process and assumes the NYT initially contain the set of all integers.

**2.7.1 δ-Huffman (n+1).** The $\delta$-Huffman (n+1) uses the feature that for every integer $n > 1$, the Elias Delta ($\delta$) code $n$ ($\delta(n)$) has a most significant bit (MSB) of 0. When an un-encountered integer $n$ arrives, $n$ is removed from the NYT list and placed into the AT list with a weight value of 1. The encoder transmits the Elias Delta code for the new value $n$, $< \delta(n + 1) >$, to the decoder. Then, with the new AT value, the encoder and decoder update their Huffman trees.

When an encountered integer $n$ arrives, it's current Huffman code and a flag bit of '1,' that is $< 1, H(n) >$, is sent to the decoder. Next, the weight value of $n$ in the AT list increases by 1. Finally, with the new AT value, the encoder and decoder update their Huffman trees.

**2.7.2 δ-Huffman (Flag/Fixed Length Coding(FLC)).** The $\delta$-Huffman (Flag/Fixed Length Coding (FLC)) assumes integers are bounded. When an un-encountered integer $n$ arrives, $n$ is removed from the NYT list and placed into the AT list with a weight value of 1. The encoder transmits a flag bit '0' and the FLC of $n$, that is $< 0, FLC(n) >$, to the decoder. Then, with the new AT value, the encoder and decoder update their Huffman trees.

When an encountered integer $n$ arrives, its current Huffman code and a flag bit of '1,' that is $< 1, H(n) >$, is sent to the decoder. Next, the weight value of $n$ in the AT list increases by 1. Finally, with the new AT value, the encoder and decoder update their Huffman trees.

**2.7.3 δ-Huffman (Reconstruction with an Exception Code).** The $\delta$-Huffman (Reconstruction with an Exception Code) assumes, upon the arrival of an un-encountered integer $n$, $n$ is removed from the NYT and placed into the AT list with a weight of 1. The value of the path to a special symbol node that represents the NYT in the Huffman tree is the "Huffman code for the NYT" $(H(NYT))$ and it maintains a weight of 0. This value along with the Elias Delta code of $n$, that is $< H(NYT), \delta(n) >$ is transmitted to the decoder. Then, with the updated AT, the current encoder and decoder Huffman trees are efficiently updated.

When an encountered integer $n$ arrives, its current Huffman code, $< H(N) >$, is sent to the decoder. Next, the weight value of $n$ in the AT list increases by 1. Finally, with the new AT value, the encoder and decoder update their Huffman trees.

**2.7.4 δ-Huffman (Update using the Sibling Property with an Exception Code).** The $\delta$-Huffman (Update using the Sibling Property with an Exception Code) assumes, upon the arrival of an un-encountered integer $n$, $n$ is removed from the NYT and placed into the AT list with a weight of 1. The Huffman code of the NYT and the Elias Delta code of $n$, that is $< H(NYT), \delta(n) >$, is transmitted to the decoder. Next, with the updated AT, the current encoder and decoder Huffman trees are efficiently updated via the sibling property [6].

When an encountered integer $n$ arrives, its current Huffman code, $< H(n) >$, is sent to the decoder. Next, the weight value of $n$ in the AT list increases by 1. Finally, with the new AT value, the encoder and decoder update their Huffman trees.

**2.7.5 δ-Huffman (Static Probability).** The $\delta$-Huffman (Static Probability) assumes the $H(NYT)$ initially starts with a weight of 1. Upon the arrival of an un-encountered integer $n$, $n$ is removed from the NYT and placed into the AT list with a weight value that is determined by a GPMF with probability value of 0.01 if $n$ is from GPMF data or a Poisson probability mass function with λ = 128 if it is from Poisson data. The Huffman code of the NYT, along with the Elias Delta code of $n$, that is $< H(NYT), \delta(n) >$ is transmitted to the decoder. Next, the weight value of NYT is updated by subtracting the weight of $n$ from the weight of the NYT. Then, with the updated AT, the current encoder and decoder Huffman trees are efficiently updated.

When an encountered integer $n$ arrives, its current Huffman code, $< H(n) >$, is sent to the decoder. No updates are made to the AT list or to the Huffman trees of the encoder and decoder.

**2.7.6 δ-Huffman (Sibling/Static Probability).** The $\delta$-Huffman (Sibling/Static Probability) makes the assumption in the estimation of the probability function to use a static probability of 0.01 with a GPMF for all data sets, with the exception of λ = 128 for data sets with a Poisson probability mass function.

Initially, the NYT node in the Huffman tree has a weight value of 1. When an un-encountered value $n$ from an input source arrives at the encoder, $n$ is removed from the NYT and placed into the AT list. Next, the encoder transmits the $H(NYT)$ code and the Elias Delta code of $n$, that is $< H(NYT), \delta(n) >$, to the decoder. Then both the encoder and decoder generate a probability value for the new value $n$ with the use of the GPMF with a static probability of 0.01, or in the case that $n$ belongs to a Poisson probability mass function, the use of λ = 128 with a Poisson probability mass function. Then the current weight of the NYT node decreases by the same amount as the probability value of the new value $n$. Finally, both the encoder and decoder update their Huffman trees and keep the sibling property.

When an encountered value $n$ arrives at the encoder that is already in the AT list, its current Huffman code, $< H(n) >$, is sent to the decoder. No updates are made to the AT list or to the Huffman trees of the encoder and decoder.

**2.7.7 δ-Huffman (Dynamic Probability P2).** The $\delta$-Huffman (Dynamic Probability P2) makes the assumption in the estimation of the probability function that the encoder and decoder dynamically update their probability values for the data $n$ as the last step in the algorithm process with the use of the dynamic probability formula P2.

Initially, the Huffman tree contains only the NYT node that starts with and maintains a weight value of zero. When an un-encountered value $n$ arrives, $n$ is removed from the NYT and placed into the AT list. The encoder transmits the $H(NYT)$ code and the Elias Delta code of $n$, that is $< H(NYT), \delta(n) >$, to the decoder. Then both the encoder and decoder update their Huffman trees with the use of the dynamic probability formula P2. The formula P2 generates probability values for all values of $n$ in the AT list. These probability values act as their weight values for the new Huffman tree.

When an encountered value $n$ arrives at the encoder that is already in the AT list, its current Huffman code, $< H(n) >$, is sent to the decoder. No updates are made to the AT list or to the Huffman trees of the encoder and decoder.

**2.7.8 δ-Huffman (Dynamic Probability P1/Slice).** The $\delta$-Huffman (Dynamic Probability P1/Slice) makes the assumption in the estimation of the probability function that the encoder and decoder dynamically update their probability values for the data $n$ as the last step in the algorithm process after a set number of iterations $(i)$ at the start of the algorithm and then after 128 iterations, the size of a slice segment $(L)$, occur with the use of the dynamic probability formula P1.

$\delta$-Huffman (Dynamic Probability P1/Slice) assumes the $H(NYT)$ initially starts with a weight value of 1. When an un-encountered value $n$ arrives, $n$'s weight value, calculated from the formula P1, is subtracted from the weight value of $H(NYT)$ in the Huffman tree. The value of $H(NYT)$ can decrease towards a minimum of zero as the algorithm runs.

For the first 16 iterations, when an un-encountered value $n$ arrives, $n$ is removed from the NYT and is placed into the AT list with a weight value determined by the formula P1 and the value of $H(NYT)$ is updated. The encoder sends nothing to the decoder. The encoder uses these first 16 data set values to learn about the data source.

At the end of iteration 16, the algorithm updates the Huffman tree with the use of formula P1 to generate probability values for all values of $n$ in the AT list. These probability values act as their weight values for the new Huffman tree. The algorithm updates the $H(NYT)$ value which the algorithm keeps for the next 16 iterations to inform the decoder that an un-encountered value had arrived.

Then the algorithm performs a loop and executes the same procedures when an un-encountered value $n$ arrives for the next 16 iterations before it restarts the loop and performs the same procedures for the next 32 iterations and then again for the next 64 iterations. At the end of the loop execution, the algorithm is at the end of iteration 128, or the size of the first slice segment, that is $L = 1$. For this reason, to explain what occurs after the first 16 iterations in the algorithm, a power value of 2 with a counter variable $t$ with a start value of 4 and end value of 6, that is $t = 4 \dots 6$ is used.

Therefore, the loop is: for $t = 4 \dots 6$ for the next $2^t$ iterations, when an un-encountered value $n$ arrives, $n$ is removed from the NYT and placed into the AT list with a weight value determined by the formula P1. The encoder transmits the $H(NYT)$ value from the end of iteration $2^t$ and the Elias Delta code of $n$, that is $< H(NYT), \delta(n) >,$ to the decoder. At the end of iteration $2^{t+1}$, the algorithm updates the Huffman tree with the use of formula P1 to generate probability values for all values of $n$ in the AT list. These probability values act as their weight values for the new Huffman tree. The algorithm updates the $H(NYT)$ value which the algorithm keeps for the next $2^{t+1}$ iterations to inform the decoder that an un-encountered value had arrived.

Then for slice numbers $L = 1 \dots \infty$, for the next 128 iterations, when an un-encountered value $n$ arrives, $n$ is removed from the NYT and placed into the AT list with a weight value determined by the formula P1. The encoder transmits the $H(NYT)$ value from the end of the previous slice $(L)$ and the Elias Delta code of $n$, that is $< H(NYT), \delta(n) >$, to the decoder. At the end of 128 iterations, the algorithm updates the Huffman tree with the use of formula P1 to generate probability values for all values in the AT list and updates the $H(NYT)$ value.

When an encountered value $n$ that belongs to AT arrives, for the first 16 iterations, the encoder sends nothing to the decoder. The encoder uses these first 16 data set values to learn about the data source. At the end of iteration 16, the algorithm updates the Huffman tree with the use of the formula P1 and updates the $H(NYT)$ value.

Then the algorithm performs a loop and executes the same procedures when an encountered value $n$ arrives for the next 16 iterations before it restarts the loop and performs the same procedures for the next 32 iterations and then again for the next 64 iterations. At the end of the loop execution, the algorithm is at the end of iteration 128, or the size of the first slice segment, that is $L = 1$. For this reason, to explain what occurs after the first 16 iterations in the algorithm, a power value of 2 with a counter variable $t$ with a start value of 4 and end value of 6, that is $t = 4 \dots 6$ is used.

Therefore, the loop is: for $t = 4 \dots 6$, in the next $2^t$ iterations, the encoder sends the Huffman code, $< H(n) >$, value for the data value $n$ at the end of iteration $2^t$. At the end of iteration $2^{t+1}$, the algorithm updates the Huffman tree with the use of the formula P1 and updates the $H(NYT)$ value.

Then for slice numbers $L = 1 \dots \infty$, for the next 128 iterations, the encoder transmits the $< H(n) >$ value from the end of the previous slice $(L)$ to the decoder. At the end of 128 iterations, the algorithm updates the Huffman tree with the use of the formula P1 and updates the $H(NYT)$ value.

**2.7.9 δ-Huffman (Dynamic Probability P2/Slice).** The $\delta$-Huffman (Dynamic Probability P2/Slice) makes the assumption in the estimation of the probability function that the encoder and decoder dynamically update their probability values for the data $n$ as the last step in the algorithm process after a set number of iterations $(i)$ at the start of the algorithm and then after 128 iterations, the size of a slice segment $(L)$, occur with the use of the dynamic probability formula P2.

$\delta$-Huffman (Dynamic Probability P2/Slice) assumes the $H(NYT)$ initially starts with and maintains a weight value of 0.

For the first 16 iterations, when an un-encountered value $n$ arrives, $n$ is removed from the NYT and placed into the AT list with a weight value determined by the formula P2 and the value of $H(NYT)$ is updated. The encoder sends nothing to the decoder. The encoder uses these first 16 data set values to learn about the data source.

At the end of iteration 16, the algorithm updates the Huffman tree with the use of formula P2 to generate probability values for all values of $n$ in the AT list. These probability values act as their weight values for the new Huffman tree. The algorithm updates the $H(NYT)$ value which the algorithm keeps for the next 16 iterations to inform the decoder that an un-encountered value had arrived.

Then the algorithm performs a loop and executes the same procedures when an un-encountered value $n$ arrives for the next 16 iterations before it restarts the loop and performs the same procedures for the next 32 iterations and then again for the next 64 iterations. At the end of the loop execution, the algorithm is at the end of iteration 128, or the size of the first slice segment, that is $L = 1$. For this reason, to explain what occurs after the first 16 iterations in the algorithm, a power value of 2 with a counter variable $t$ with a start value of 4 and end value of 6, that is $t = 4 \dots 6$ is used.

Therefore, the loop is: for $t = 4 \dots 6$ for the next $2^t$ iterations, when an un-encountered value $n$ arrives, $n$ is removed from the NYT and placed into the AT list with a weight value determined by the formula P2. The encoder transmits the $H(NYT)$ value from the end of iteration $2^t$ and the Elias Delta code of $n$, that is $< H(NYT), \delta(n) >,$ to the decoder. At the end of iteration $2^{t+1}$, the algorithm updates the Huffman tree with the use of formula P2 to generate probability values for all values of $n$ in the AT list. These probability values act as their weight values for the new Huffman tree. The algorithm updates the $H(NYT)$ value which the algorithm keeps for the next $2^{t+1}$ iterations to inform the decoder that an un-encountered value had arrived.

Then for slice numbers $L = 1 \dots \infty$, for the next 128 iterations, when an un-encountered value $n$ arrives, $n$ is removed from the NYT and placed into the AT list with a weight value determined by the formula P2. The encoder transmits the $H(NYT)$ value from the end of the previous slice $(L)$ and the Elias Delta code of $n$, that is $< H(NYT), \delta(n) >$, to the decoder. At the end of 128 iterations, the algorithm updates the Huffman tree with the use of formula P2 to generate probability values for all values in the AT list and updates the $H(NYT)$ value.

When an encountered value $n$ that belongs to AT arrives, for the first 16 iterations, the encoder sends nothing to the decoder. The encoder uses these first 16 data set values to learn about the data source. At the end of iteration 16, the algorithm updates the Huffman tree with the use of the formula P2 and updates the $H(NYT)$ value.

Then the algorithm performs a loop and executes the same procedures when an encountered value $n$ arrives for the next 16 iterations before it restarts the loop and performs the same procedures for the next 32 iterations and then again for the next 64 iterations. At the end of the loop execution, the algorithm is at the end of iteration 128, or the size of the first slice segment, that is $L = 1$. For this reason, to explain what occurs after the first 16 iterations in the algorithm, a power value of 2 with a counter variable $t$ with a start value of 4 and end value of 6, that is $t = 4 \dots 6$ is used.

Therefore, the loop is: for $t = 4 \dots 6$, in the next $2^t$ iterations, the encoder sends the Huffman code, $< H(n) >$, value for the data value $n$ at the end of iteration $2^t$. At the end of iteration $2^{t+1}$, the algorithm updates the Huffman tree with the use of the formula P2 and updates the $H(NYT)$ value.

Then for slice numbers $L = 1 \dots \infty$, for the next 128 iterations, the encoder transmits the $< H(n) >$ value from the end of slice $(L)$ to the decoder. At the end of 128 iterations, the algorithm updates the Huffman tree with the use of the formula P2 and updates the $H(NYT)$ value.

**2.7.10 δ-Huffman (Dynamic Probability P1/Slice/Reset).** The $\delta$-Huffman (Dynamic Probability P1/Slice/Reset) makes the assumption in the estimation of the probability function that the encoder and decoder dynamically update their probability values for the data $n$ as the last step in the algorithm process after a set number of iterations $(i)$ at the start of the algorithm and then after 128 iterations, the size of a slice segment $(L)$, occur with the use of the dynamic probability formula P1. For $L = 1 \dots \infty$, after the Huffman tree is updated, this algorithm resets the $\sum_0^i n$ value in the dynamic probability formula P1 to zero and the NYT weight value back to one.

$\delta$-Huffman (Dynamic Probability P1/Slice/Reset) assumes the $H(NYT)$ initially starts with a weight of 1. When an un-encountered value $n$ arrives, $n$'s weight value, calculated from the formula P1, is subtracted from the weight value of $H(NYT)$ in the Huffman tree. The value of $H(NYT)$ can decrease towards a minimum of zero as the algorithm runs.

For the first 16 iterations, when an un-encountered value $n$ arrives, $n$ is removed from the NYT and placed into the AT list with a weight value determined by the formula P1 and the value of $H(NYT)$ is updated. The encoder sends nothing to the decoder. The encoder uses these first 16 data set values to learn about the data source.

At the end of iteration 16, the algorithm updates the Huffman tree with the use of formula P1 to generate probability values for all values of $n$ in the AT list. These probability values act as their weight values for the new Huffman tree. The algorithm updates the $H(NYT)$ value which the algorithm keeps for the next 16 iterations to inform the decoder that an un-encountered value had arrived.

Then the algorithm performs a loop and executes the same procedures when an un-encountered value $n$ arrives for the next 16 iterations before it restarts the loop and performs the same procedures for the next 32 iterations and then again for the next 64 iterations. At the end of the loop execution, the algorithm is at the end of iteration 128, or the size of the first slice segment, that is $L = 1$. For this reason, to explain what occurs after the first 16 iterations in the algorithm, a power value of 2 with a counter variable $t$ with a start value of 4 and end value of 6, that is $t = 4 \dots 6$ is used.

Therefore, the loop is: for $t = 4 \dots 6$ for the next $2^t$ iterations, when an un-encountered value $n$ arrives, $n$ is removed from the NYT and placed into the AT list with a weight value determined by the formula P1. The encoder transmits the $H(NYT)$ value from the end of iteration $2^t$ and the Elias Delta code of $n$, that is $< H(NYT), \delta(n) >,$ to the decoder. At the end of iteration $2^{t+1}$, the algorithm updates the Huffman tree with the use of formula P1 to generate probability values for all values of $n$ in the AT list. These probability values act as their weight values for the new Huffman tree. The algorithm updates the $H(NYT)$ value which the algorithm keeps for the next $2^{t+1}$ iterations to inform the decoder that an un-encountered value had arrived.

Then for slice numbers $L = 1 \ldots \infty$, for the next 128 iterations, when an un-encountered value $n$ arrives, $n$ is removed from the NYT and placed into the AT list with a weight value determined by the formula P1. The encoder transmits the $H(NYT)$ value from the end of the previous slice $(L)$ and the Elias Delta code of $n$, that is $< H(NYT), \delta(n) >$, to the decoder. At the end of 128 iterations, the algorithm updates the Huffman tree with the use of formula P1 to generate probability values for all values in the AT list and updates the $H(NYT)$ value. Then the algorithm resets the $\sum_0^i n$ value in the formula P1 to zero and the NYT weight value back to one.

When an encountered value $n$ that belongs to AT arrives, for the first 16 iterations, the encoder sends nothing to the decoder. The encoder uses these first 16 data set values to learn about the data source. At the end of iteration 16, the algorithm updates the Huffman tree with the use of the formula P1 and updates the $H(NYT)$ value.

Then the algorithm performs a loop and executes the same procedures when an encountered value $n$ arrives for the next 16 iterations before it restarts the loop and performs the same procedures for the next 32 iterations and then again for the next 64 iterations. At the end of the loop execution, the algorithm is at the end of iteration 128, or the size of the first slice segment, that is $L = 1$. For this reason, to explain what occurs after the first 16 iterations in the algorithm, a power value of 2 with a counter variable $t$ with a start value of 4 and end value of 6, that is $t = 4 \ldots 6$ is used.

Therefore, the loop is: for $t = 4 \ldots 6$, in the next $2^t$ iterations, the encoder sends the Huffman code, $< H(n) >$, value for the data value $n$ at the end of iteration $2^t$. At the end of iteration $2^{t+1}$, the algorithm updates the Huffman tree with the use of the formula P1 and updates the $H(NYT)$ value.

Then for slice numbers $L = 1 \ldots \infty$, for the next 128 iterations, the encoder transmits the $< H(n) >$ value from the end of the previous slice $(L)$ to the decoder. At the end of 128 iterations, the algorithm updates the Huffman tree with the use of the formula P1 and updates the $H(NYT)$ value. Then the algorithm resets the $\sum_0^i n$ value in the formula P1 to zero and the NYT weight value back to one.

**2.7.11 δ-Huffman (Dynamic Probability P2/Slice/Reset).** The $\delta$-Huffman (Dynamic Probability P2/Slice/Reset) makes the assumption in the estimation of the probability function that the encoder and decoder dynamically update their probability values for the data $n$ as the last step in the algorithm process after a set number of iterations $(i)$ at the start of the algorithm and then after 128 iterations, the size of a slice segment $(L)$, occur with the use of the dynamic probability formula P2. For $L = 1 \ldots \infty$, after the Huffman tree is updated, this algorithm resets the weight values for all integers currently in the AT list to 0.

$\delta$-Huffman (Dynamic Probability P2/Slice/Reset) assumes the $H(NYT)$ initially starts with and maintains a weight value of 0.

For the first 16 iterations, when an un-encountered value $n$ arrives, $n$ is removed from the NYT and placed into the AT list with a weight value determined by the formula P2 and the value of $H(NYT)$ is updated. The encoder sends nothing to the decoder. The encoder uses these first 16 data values to learn about the data source.

At the end of iteration 16, the algorithm updates the Huffman tree with the use of formula P2 to generate probability values for all values of $n$ in the AT list. These probability values act as their weight values for the new Huffman tree. The algorithm updates the $H(NYT)$ value which the algorithm keeps for the next 16 iterations to inform the decoder that an un-encountered value had arrived.

Then the algorithm performs a loop and executes the same procedures when an un-encountered value $n$ arrives for the next 16 iterations before it restarts the loop and performs the same procedures for the next 32 iterations and then again for the next 64 iterations. At the end of the loop execution, the algorithm is at the end of iteration 128, or the size of the first slice segment, that is $L = 1$. For this reason, to explain what occurs after the first 16 iterations in the algorithm, a power value of 2 with a counter variable $t$ with a start value of 4 and end value of 6, that is $t = 4 \dots 6$ is used.

Therefore, the loop is: for $t = 4 \dots 6$ for the next $2^t$ iterations, when an un-encountered value $n$ arrives, $n$ is removed from the NYT and placed into the AT list with a weight value determined by the formula P2. The encoder transmits the $H(NYT)$ value from the end of iteration $2^t$ and the Elias Delta code of $n$, that is $< H(NYT), \delta(n) >,$ to the decoder. At the end of iteration $2^{t+1}$, the algorithm updates the Huffman tree with the use of formula P2 to generate probability values for all values of $n$ in the AT list. These probability values act as their weight values for the new Huffman tree. The algorithm updates the $H(NYT)$ value which the algorithm keeps for the next $2^{t+1}$ iterations to inform the decoder that an un-encountered value had arrived.

Then for slice numbers $L = 1 \ldots \infty$, for the next 128 iterations, when an un-encountered value $n$ arrives, $n$ is removed from the NYT and placed into the AT list with a weight value determined by the formula P2. The encoder transmits the $H(NYT)$ value from the end of the previous slice $(L)$ and the Elias Delta code of $n$, that is $< H(NYT), \delta(n) >$, to the decoder. At the end of 128 iterations, the algorithm updates the Huffman tree with the use of formula P2 to generate probability values for all values in the AT list and updates the $H(NYT)$ value. Then the algorithm resets the weight values for all integers currently in the AT list to 0.

When an encountered value $n$ that belongs to AT arrives, for the first 16 iterations, the encoder sends nothing to the decoder. The encoder uses these first 16 data set values to learn about the data source. At the end of iteration 16, the algorithm updates the Huffman tree with the use of the formula P2 and updates the $H(NYT)$ value.

Then the algorithm performs a loop and executes the same procedures when an encountered value $n$ arrives for the next 16 iterations before it restarts the loop and performs the same procedures for the next 32 iterations and then again for the next 64 iterations. At the end of the loop execution, the algorithm is at the end of iteration 128, or the size of the first slice segment, that is $L = 1$. For this reason, to explain what occurs after the first 16 iterations in the algorithm, a power value of 2 with a counter variable $t$ with a start value of 4 and end value of 6, that is $t = 4 \ldots 6$ is used.

Therefore, the loop is: for $t = 4 \ldots 6$ for the next $2^t$ iterations, the encoder sends the Huffman code, $< H(n) >$, value for the data value $n$ at the end of iteration $2^t$. At the end of iteration $2^{t+1}$, the algorithm updates the Huffman tree with the use of the formula P2 and updates the $H(NYT)$ value.

Then for slice numbers $L = 1 \dots \infty$, for the next 128 iterations, the encoder transmits the $< H(n) >$ value from the end of slice $(L)$ to the decoder. At the end of 128 iterations, the algorithm updates the Huffman tree with the use of the formula P2 and updates the $H(NYT)$ value. Then the algorithm resets the weight values for all integers currently in the AT list to 0.

**2.7.12 δ-Huffman (Fixed Length Coding).** The $\delta$-Huffman (FLC) assumes the integers are bounded and the $H(NYT)$ starts with and maintains a weight value of 0. When a new integer $n$ arrives, $n$ is removed from the NYT list and placed into the AT list with a weight value of 1. The encoder transmits the $H(NYT)$, and the FLC of $n$, that is $< H(NYT), FLC(n) >$, to the decoder. Then, with the new AT value, the encoder and decoder update their Huffman trees.

When an integer $n$ that is already in the AT list arrives, its current Huffman code, $< H(n) >$, is sent to the decoder. Next, the weight value of $n$ in the AT list increases by 1. Finally, with the new AT value, the encoder and decoder update their Huffman trees.

# 3. LITERATURE REVIEW

This chapter compares the work done in this thesis to the methods and techniques of compression of bounded and unbounded integers performed by other researchers. To the best of this writer's knowledge, this is the first work that uses the reported approach for estimating the probability mass function for the $\delta$-Huffman algorithm.

Orlitsky et al. dealt with patterns and how to use them in compression algorithms that could work on alphabets of unbounded size [8], and Merhav et al. works with two-sided geometric distributions [9]. This work replaces the emphasis from patterns or two-sided geometric distributions to the $\delta$-Huffman.

Gallager et al. [10] Kato et al. [11] and Abrahams et al. [12] assume that the integers have a specific type of probability mass function in their works of interest. This paper works with the $\delta$-Huffman algorithm with different assumptions about the probability mass function.

Uyematsu and Kanaya work updates the LZ77 and LZ78 algorithms to fit unbounded integers [13]. This work replaces the emphasis from the LZ77 and LZ78 algorithms to that of the $\delta$-Huffman.

"Universal Lossless Coding of Sources with Large and Unbounded Alphabets" by En-hui Yang and Yunwei Jia [14], consider the general case, where the alphabet may increase without bound, and the decoder does not know how it grows. To encode such a data sequence $X = x_1 x_2 \dots x_n$, they combine Elias coding with a dynamically updated binary search tree. The proposed algorithm works as follows: For each symbol $x_i$ in the input sequence, if it has not appeared before in $x_1 \dots x_{i-1}$ , use the Elias code to encode $x_i$ and then add this symbol to the corresponding leaf sub-alphabet and update the tree structure; if $x_i$ has appeared before, then encode the corresponding path in the dynamic tree and the index in the corresponding leaf sub-alphabet. We use the $\delta$-Huffman algorithm to handle unbounded integers.

“Dynamic Unbounded Integer Compression” by Naga Sai Gowtham Mulpuri [15] explores the possibility of combining integer compression methods with a new dynamic Huffman compression algorithm known as $\delta$-Huffman. It shows the bit rate values of the $\delta$-Huffman algorithm to be very close to the entropy of the data source and that 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. Additionally, [15] compares the bit rate of δ-Huffman to the bit rate of the compression techniques “gzip” and “bzip2.” The work in [15] mainly explores the achievable compression ratios in the context of $\delta$-Huffman encoding.

“Adaptive Single-Pass Compression of Unbounded Integers” by Hyatt, Christopher Rice [16] explores the ability to dynamically compress data in one pass with Variable length nibbles with Tunstall (VLNT) and Delta-Tunstall ($\delta$-T).

The main difference among our current work from [15] and [16] is that we focus on different assumptions about the estimation of the probability function for the $\delta$-Huffman algorithm. Inclusive for this work is that all $\delta$-Huffman algorithms are examined prior to comparison to [15] which is provided in Chapter 6.

# 4. EXPERIMENTAL SETUP

This chapter discusses the setup methodology and summarizes the twelve experiments. Chapters 5 and 6 discuss the results of these experiments.

The experiments use one or more of three data sets: synthetic data, real-world data from sorted inverted index gaps from Wikipedia, and benchmark data from Silesia that attempts to represent realistic workload data. The entropy estimates of these data sets and the average bit rate of the twelve $\delta$-Huffman algorithms offer a way to construct a comparative evolution of the performance of the $\delta$-Huffman algorithms.

## 4.1 Summary of Input Sources

This section discusses the input sources and their general characteristics.

**4.1.1 Synthetic Data.** The synthetic data derives from two general probability mass functions: GPMF and Poisson. Four different data sets make up the synthetic input data: Data GPMF 0.01, Data GPMF 0.1, Data GPMF 0.5, and Poisson. Each of the inputs consists of positive integers that distribute with different probabilities. For the GPMF inputs, the probabilities are 0.01, 0.1, and 0.5. The Poisson input uses a $\lambda = 128$.

- "Data GPMF 0.01" contains 10,000 integers. The minimum integer value is 1 and maximum integer value is 826.
- "Data GPMF 0.1" contains 10,000 integers. The minimum integer value is 1 and maximum integer value is 99.
- "Data GPMF 0.5" contains 10,000 integers. The minimum integer value is 1 and maximum integer value is 14.
- "Poisson" contains 10,000 integers. The minimum integer value is 89 and maximum integer value is 147.

**4.1.2 Wikipedia Gap Sources.** The Wikipedia gap sources are from the sorted inverted indexes for several common search terms from Wikipedia at the end of the year 2015. The search terms the experiments use are “2015”, “Bollywood”, “Grei”, “Rousei”, and “State.”

- “2015” contains 324,888 integers. The minimum integer value is 1 and maximum is 75,130.
- “Bollywood” contains 10,605 integers. The minimum integer value is 1 and maximum is 474,315.
- “Grei” contains 49,973 integers. The minimum integer value is 1 and maximum is 421,836.
- “Rousei” is the smallest data set and contains 211 integers. The minimum integer value is 754 and maximum is 1,932,275.
- “State” is the biggest data set and contains 1,237,789 integers. The minimum integer value is 1 and maximum is 37,064.

**4.1.3 Silesia Sources.** The Silesia sources are a standard data set that can compare compression algorithms. The data set is made up of 12 different data sets made up of many different formats of data [17].

- “dickens” contains the text of the works of Charles Dickens. The total number of bytes is 10,192,446. The minimum value is 9 and maximum is 129.
- “mozilla” contains the tarred executables of Mozilla 1.0. The total number of bytes is 51,220,480. The minimum value is 0 and maximum is 255.
- “mr” contains a collection of MRI images. The total number of bytes is 9,970,564. The minimum value is 0 and maximum value is 255.

- “nci” contains a database of chemical structures. The total number of bytes is 33,553,445. The minimum value is 10 and maximum value is 118.
- “ooffice” contains the dll from Open Office 1.01. The total number of bytes is 6,152,192. The minimum value is 0 and maximum value is 255.
- “osdb” contains the Open Source Database Benchmark (osdb) in the MySQL format. The total number of bytes is 10,085,684. The minimum value is 0 and maximum value is 255.
- “reymont” contains the book Chlopi by Wladyslaw Reymont. The total number of bytes is 6,627,202. The minimum value is 0 and maximum value is 255.
- “samba” contains the source code of Samba 2-2.3. The total number of bytes is 21,606,400. The minimum value is 0 and maximum value is 255.
- “sao” contains the SAO star catalog in the form of bin data. The total number of bytes is 7,251,944. The minimum value is 0 and maximum value is 255.
- “webster” contains the 1913 unabridged dictionary text. The total number of bytes is 41,458,703. The minimum value is 10 and maximum value is 126.
- “xml” contains a collection of XML files. The total number of bytes is 5,345,280. The minimum value is 0 and maximum value is 252.
- “x-ray” contains a collection of x-ray pictures. The total number of bytes is 8,474,240. The minimum value is 0 and maximum value is 255.

## 4.2 Experiment Summaries

This section summarizes each of the twelve experiments. To allow comparisons among the different $\delta$-Huffman variants, experiments that use the benchmark data set Silesia do so as a data set of integers.

**4.2.1 Experiment 1:** Experiment 1 evaluates the $\delta$-Huffman (n+1) assumptions for the estimation of the probability function. The average bit rate results are compared to the entropy of the synthetic data sets of 10,000 integers that use a GPMF with probability values 0.5, 0.1, 0.01, and a Poisson probability mass function, with $\lambda = 128$.

**4.2.2 Experiment 2:** Experiment 2 evaluates the $\delta$-Huffman (Flag/FLC) assumptions for the estimation of the probability function. The average bit rate results are compared to the entropy of the synthetic data sets of 10,000 integers, that this algorithm handles as a data set of bytes, that use a GPMF with probability values 0.5, 0.1, 0.01, and a Poisson probability mass function, with $\lambda = 128$.

**4.2.3 Experiment 3:** Experiment 3 evaluates the $\delta$-Huffman (Reconstruction with an Exception Code) assumptions for the estimation of the probability function. The average bit rate results are compared to the entropy of the synthetic data sets for both 10,000 integers and 10,000 integers, that this algorithm handles as a data set of bytes, of GPMF with probability values 0.5, 0.1, 0.01, and a Poisson probability mass function, with $\lambda = 128$. The average bit rate results are also compared to the entropy of the benchmark data set Silesia.

**4.2.4 Experiment 4:** Experiment 4 evaluates the $\delta$-Huffman (Update using the Sibling Property with an Exception Code) assumptions for the estimation of the probability function. The average bit rate results are compared to the entropy of the synthetic data sets of 10,000 integers that use a GPMF with probability values 0.5, 0.1, 0.01, and a Poisson probability mass function, with λ = 128. The average bit rate results are also compared to the entropy of the real-world data from the sorted inverted index gaps from Wikipedia.

**4.2.5 Experiment 5:** Experiment 5 evaluates the $\delta$-Huffman (Static Probability) assumptions for the estimation of the probability function. The average bit rate results are compared to the entropy of the synthetic data sets of 10,000 integers that use a GPMF with probability values 0.5, 0.1, 0.01, and a Poisson probability mass function, with λ = 128. The average bit rate results are also compared to the entropy of the real-world data from the sorted inverted index gaps from Wikipedia.

**4.2.6 Experiment 6:** Experiment 6 evaluates the $\delta$-Huffman (Sibling/Static Probability) assumptions for the estimation of the probability function. The average bit rate results are compared to the entropy of the synthetic data sets of 10,000 integers that use a GPMF with probability values 0.5, 0.1, 0.01, and a Poisson probability mass function, with λ = 128. The average bit rate results are also compared to the entropy of the real-world data from the sorted inverted index gaps from Wikipedia.

**4.2.7 Experiment 7:** Experiment 7 evaluates the $\delta$-Huffman (Dynamic Probability P2) assumptions for the estimation of the probability function. The average bit rate results are compared to the entropy of the synthetic data sets of 10,000 integers that use a GPMF with probability values 0.5, 0.1, 0.01, and a Poisson probability mass function, with λ = 128. The average bit rate results are also compared to the entropy of the real-world data from the sorted inverted index gaps from Wikipedia and the benchmark data set Silesia.

**4.2.8 Experiment 8:** Experiment 8 evaluates the $\delta$-Huffman (Dynamic Probability P1/Slice) assumptions for the estimation of the probability function. The average bit rate results are compared to the entropy of the synthetic data sets of 10,000 integers that use a GPMF with probability values 0.5, 0.1, 0.01, and a Poisson probability mass function, with λ = 128. The average bit rate results are also compared to the entropy of the real-world data from the sorted inverted index gaps from Wikipedia and the benchmark data set Silesia.

**4.2.9 Experiment 9:** Experiment 9 evaluates the $\delta$-Huffman (Dynamic Probability P2/Slice) assumptions for the estimation of the probability function. The average bit rate results are compared to the entropy of the synthetic data sets of 10,000 integers that use a GPMF with probability values 0.5, 0.1, 0.01, and a Poisson probability mass function, with λ = 128. The average bit rate results are also compared to the entropy of the real-world data from the sorted inverted index gaps from Wikipedia and the benchmark data set Silesia.

**4.2.10 Experiment 10:** Experiment 10 evaluates the $\delta$-Huffman (Dynamic Probability P1/Slice/Reset) assumptions for the estimation of the probability function. The average bit rate results are compared to the entropy of the synthetic data sets of 10,000 integers that use a GPMF with probability values 0.5, 0.1, 0.01, and a Poisson probability mass function, with λ = 128. The average bit rate results are also compared to the entropy of the real-world data from the sorted inverted index gaps from Wikipedia and the benchmark data set Silesia.

**4.2.11 Experiment 11:** Experiment 11 evaluates the $\delta$-Huffman (Dynamic Probability P2/Slice/Reset) assumptions for the estimation of the probability function.

The average bit rate results are compared to the entropy of the synthetic data sets of 10,000 integers that use a GPMF with probability values 0.5, 0.1, 0.01, and a Poisson probability mass function, with λ = 128. The average bit rate results are also compared to the entropy of the real-world data from the sorted inverted index gaps from Wikipedia and the benchmark data set Silesia.

**4.2.12 Experiment 12:** Experiment 12 evaluates the $\delta$-Huffman (FLC) assumptions for the estimation of the probability function. The average bit rate results are compared to the entropy of the synthetic data sets of 10,000 integers, that this algorithm handles as a data set of bytes, that use a GPMF with probability values 0.5, 0.1, 0.01, and a Poisson probability mass function, with λ = 128. The average bit rate results are also compared to the entropy of the real-world data from the sorted inverted index gaps that this algorithm also handles as a data set of bytes, from Wikipedia, and the benchmark data set Silesia.

# 5. EXPERIMENTAL RESULTS

This chapter discusses the results of the twelve experiments. In the context of this paper the term local observations refers to observations made based on experimental values in the current experiment being evaluated and experimental values in prior experiments, if there are any, while the term global observations refers to observations made based on all experimental values. This chapter makes only local observations, so the bit rate values for the experiments are only shown to the second decimal place. However, Chapter 6 makes global observations for the experiments in this chapter and to show the differences between some of the δ-Huffman bit rate value results; Chapter 6 expands the bit rate value results beyond the second decimal place.

## 5.1 Experiment 1: δ-Huffman (n+1) compression of synthetic data (integers)

$\delta$-Huffman (n+1) estimates the probability function based on the use of the formula $p_n$ at the end of an iteration. Figure 5.1 compares the $\delta$-Huffman (n+1) bit rate value results to the entropy values of the data set.

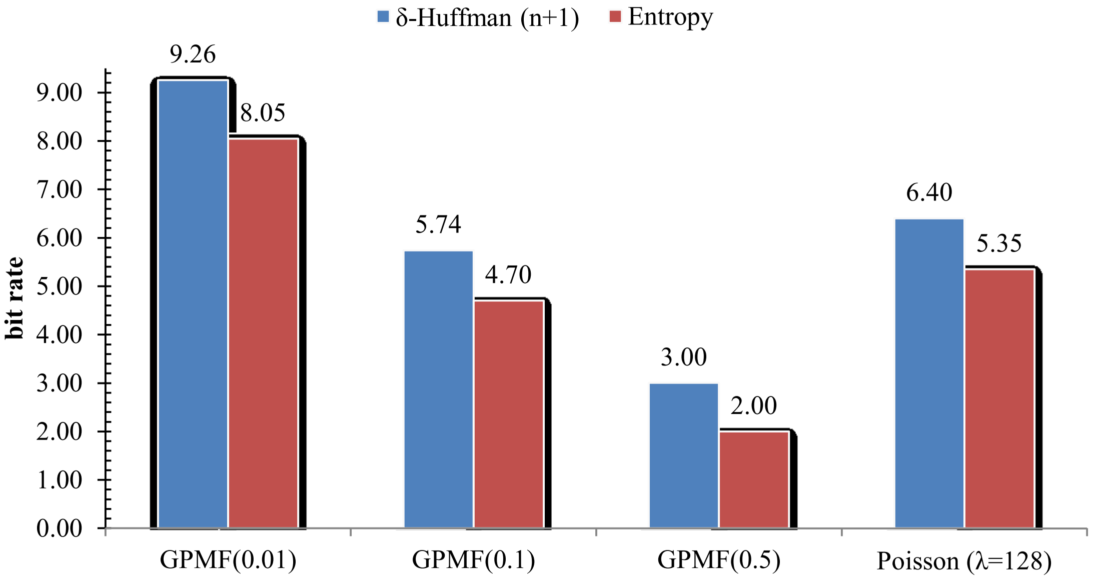


**Figure 5.1: δ-Huffman (n+1) bit rate values vs. Entropy**

From figure 5.1 it can be noted that, for $\delta$-Huffman (n+1), the bit rate values differ from their respective entropy values from a low difference of 1.00-bit for GPMF (0.5) to a higher difference of 1.21 bits for GPMF (0.01).

From this experiment it can be noted that the $\delta$-Huffman (n+1) generates bit rate values that are higher than the entropy; 1.21 bits more for GPMF (0.01), 1.04 bits more for GPMF (0.1), 1.00-bit more for GPMF (0.5), and 1.05 bits more for Poisson (λ=128) relative to the entropy of the synthetic data set of integers.

The final note for experiment 1 is that a plausible explanation for these results is that the $\delta$-Huffman code provides a lower bit rate in data sets when numerous repetitions of small integers are involved because of the repeated use of those integers from the Huffman tree which are already updated and requires fewer bits to encode than that of a new value from the data sets. In this case:

- The GPMF (0.5) data set contains very small integers with significant amount of repetitions. The maximum input value is 14 and the minimum is 1.
- The GPMF (0.1) data set contains medium small integers with numerous repetitions. The maximum input value is 99 and minimum is 1.
- The GPMF (0.01) data set contains medium small to medium large integers with numerous repetitions. The maximum input value is 826 and minimum is 1.
- The Poisson data set contains medium to medium large integers with numerous repetitions. The maximum input value is 147 and minimum is 89.

This allows the $\delta$-Huffman (n+1) to generate the highest bit rate value relative to entropy for GPMF (0.01) and the lowest bit rate value relative to entropy for GPMF (0.05).

### 5.2 Experiment 2: δ-Huffman (Flag/FLC) compression of synthetic data (bytes)

$\delta$-Huffman (Flag/FLC) estimates the probability function based on the use of the formula $p_n$ at the end of an iteration. Figure 5.2 compares the $\delta$-Huffman (Flag/FLC) bit rate values to the entropy values of the data set.

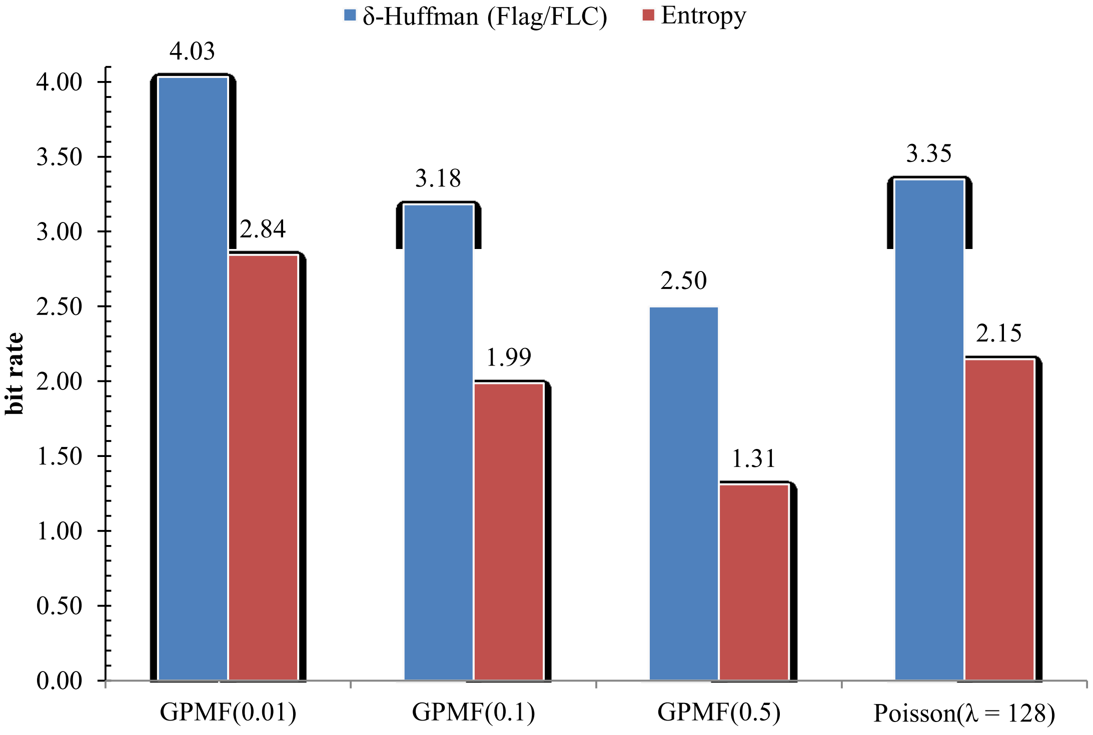


**Figure 5.2: δ-Huffman (Flag/FLC) bit rate values (bytes) vs. Entropy**

From figure 5.2 it can be noted that, for $\delta$-Huffman (Flag/FLC), the bit rate values differ from their respective entropy values from a low difference of 1.19 bits for GPMF (0.5) to a high difference of 1.20 bits for GPMF (0.1).

From this experiment it can be noted that the $\delta$-Huffman (Flag/FLC) generates bit rate values that are higher than the entropy; 1.19 bits more for GPMF (0.01), 1.20 bits more for GPMF (0.1), 1.19 bits more for GPMF (0.5), and 1.20 bits more for Poisson (λ=128) relative to the entropy of the synthetic data set of integers handled as bytes in this experiment.

The final note for experiment 2 is that the higher bit rate values relative to entropy are expected for this experiment, based on the results from experiment 1 and from the issue of the use of fixed length coding, in that the code uses the same number of bits to represent each symbol from the data source, but it is addressing the worst case scenario and does not enable compression via assignment of a smaller length code to the most probable symbols.

### 5.3 Experiment 3: δ-Huffman (Reconstruction with an Exception Code)

Experiment 3 is divided into three parts: part 3a explains the compression of the synthetic data (integers), part 3b explains the compression of the synthetic data (bytes), and part 3c explains the compression of the benchmark data set Silesia.

**5.3.1 Experiment 3a: Synthetic Data Set (integers).** $\delta$-Huffman (Reconstruction with an Exception Code) estimates the probability function based on the use of the formula $p_n$ and the use of a special symbol that represents the NYT and maintains a value of 0. Figure 5.3 compares the $\delta$-Huffman (Reconstruction with an Exception Code) bit rate values to the entropy values of the data set.

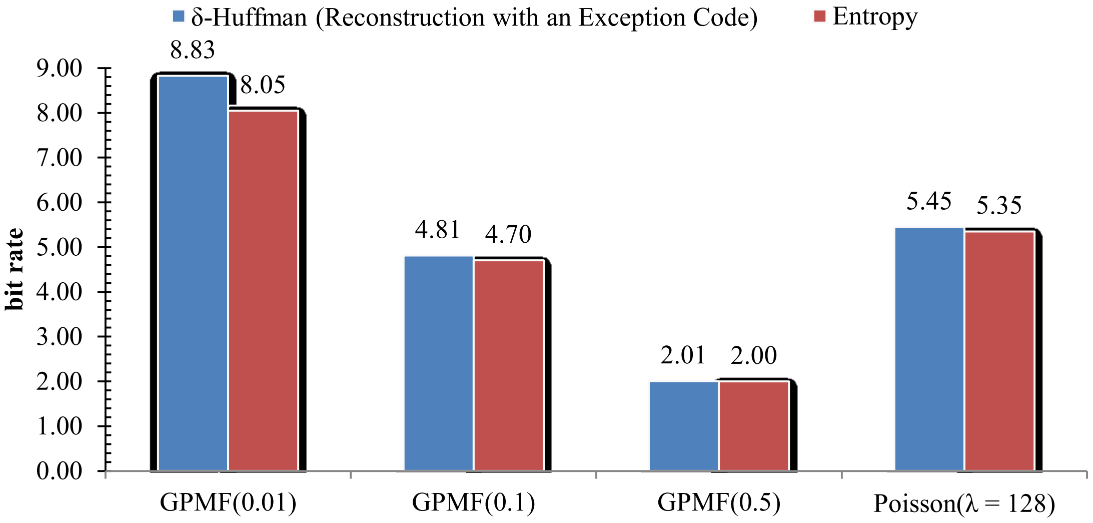


**Figure 5.3: δ-Huffman (Reconstruction with an Exception Code) bit rate values vs. Entropy**

From figure 5.3 it can be noted that, for $\delta$-Huffman (Reconstruction with an Exception Code), the bit rate values differ from their respective entropy values from a low difference of 0.01-bit for GPMF (0.5) to a high difference of 0.78-bit for GPMF (0.01).

From this experiment it can be noted that the $\delta$-Huffman (Reconstruction with an Exception Code) generates bit rate values that are higher than the entropy; 0.78-bit more for GPMF (0.01), 0.11-bit more for GPMF (0.1), 0.01-bit more for GPMF (0.5), and 0.10-bit more for Poisson (λ=128) relative to the entropy of the synthetic data set of integers.

It can be noted that the $\delta$-Huffman (Reconstruction with an Exception Code) generates lower bit rate values; 0.43-bit fewer for GPMF (0.01), 0.93-bit fewer for GPMF (0.1), 0.99-bit fewer for GPMF (0.5), and 0.96-bit fewer for Poisson (λ=128) versus δ-Huffman (n+1).

The final note for experiment 3a is that the $\delta$-Huffman (Reconstruction with an Exception Code) use of a special symbol, that represents the NYT and maintains a value of 0, improves its ability to keep un-encountered values $n$ that are seen by the encoder to enter the Huffman tree nearer the bottom of the Huffman tree where values with fewer occurrences and thus longer code-words are kept. Encountered values $n$ that have been seen more often in the input data stream are preferably towards the top of the Huffman tree, closer to the root node, with shorter code-words that are used to send to the decoder. This improvement keeps the bit rate values lower relative to entropy versus the δ-Huffman (n+1) from experiment 1.

**5.3.2 Experiment 3b: Synthetic Data Set (bytes).** Figure 5.4 compares the $\delta$-Huffman (Reconstruction with an Exception Code) bit rate values to the entropy values of the data set.

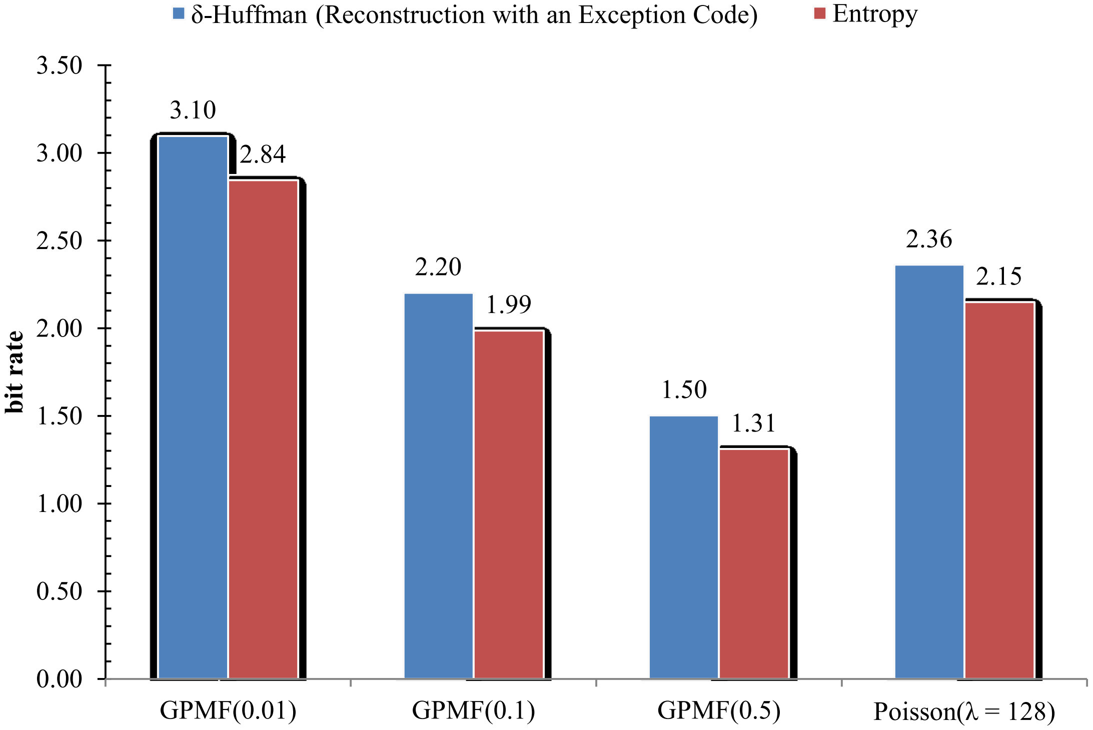


**Figure 5.4: δ-Huffman (Reconstruction with an Exception Code) bit rate values (bytes) vs. Entropy**

From figure 5.4 it can be noted that, for $\delta$-Huffman (Reconstruction with an Exception Code), the bit rate values differ from their respective entropy values from a low difference of 0.19-bit for GPMF (0.5) to a high difference of 0.25-bit for GPMF (0.01).

From this experiment it can be noted that the $\delta$-Huffman (Reconstruction with an Exception Code) generates bit rate values that are higher than the entropy; 0.25-bit more for GPMF (0.01), 0.22-bit more for GPMF (0.1), 0.19-bit more for GPMF (0.5), and 0.21-bit more for Poisson (λ=128) relative to the entropy of the synthetic data set of integers handled as bytes in this experiment.

It can be noted that the $\delta$-Huffman (Reconstruction with an Exception Code) generates lower bit rate values; 0.94-bit fewer for GPMF (0.01), 0.98-bit fewer for GPMF (0.1), 1.00-bit fewer for GPMF (0.5), and 0.99-bit fewer for Poisson (λ=128) versus $\delta$-Huffman (Flag/FLC).

The final note for experiment 3b is that the $\delta$-Huffman (Flag/FLC) variant, because of the issue with fixed length coding as discussed in experiment 3a, generates worse bit rate values relative to entropy versus the $\delta$-Huffman (Reconstruction with an Exception Code) variant which can generate smaller length code for the most probable symbols.

**5.3.3 Experiment 3c: Benchmark Data Set (Silesia).** Figure 5.5 compares the $\delta$-Huffman (Reconstruction with an Exception Code) bit rate values to the entropy values of the data set.

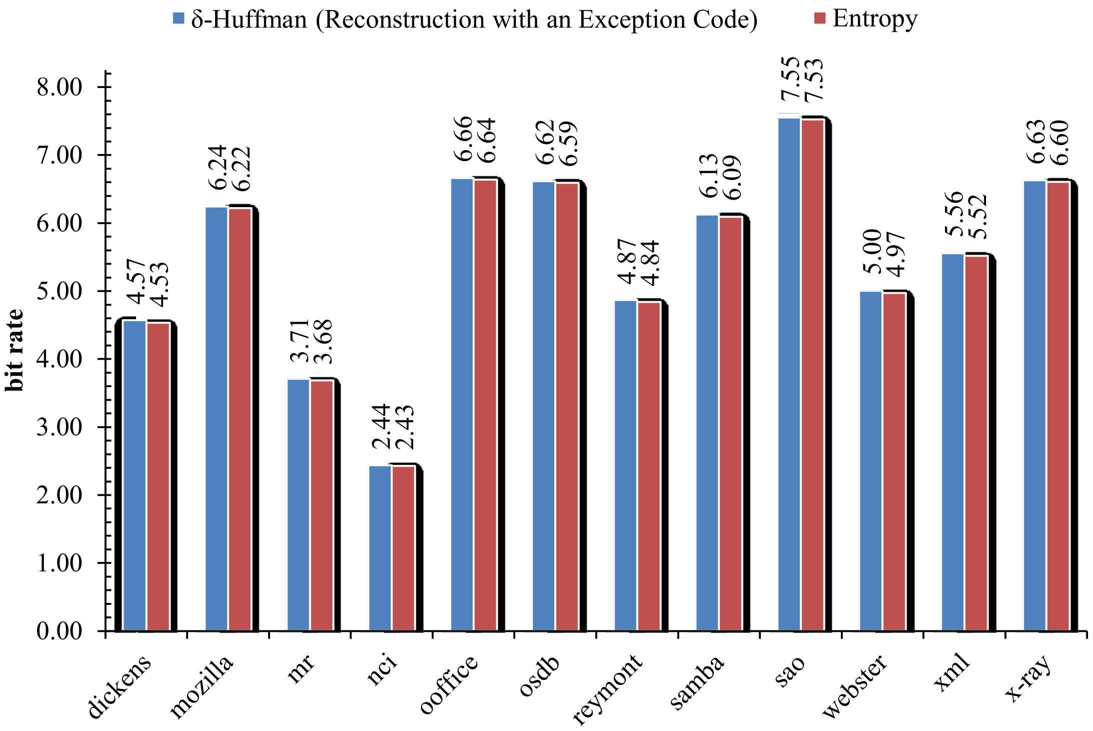


**Figure 5.5: δ-Huffman (Reconstruction with an Exception Code) bit rate values vs. Entropy**

From figure 5.5 it can be noted that, for $\delta$-Huffman (Reconstruction with an Exception Code), the bit rate values differ from their respective entropy values from a low difference of 0.01-bit for Silesia (nci) to a high difference of 0.04-bit for Silesia (dickens) and Silesia (xml).

From this experiment it can be noted that the $\delta$-Huffman (Reconstruction with an Exception Code) generates bit rate values that are higher than the entropy; 0.04-bit more for Silesia (dickens), 0.02-bit more for Silesia (mozilla), 0.03-bit more for Silesia (mr), 0.01-bit more for Silesia (nci), 0.02-bit more for Silesia (ooffice), 0.02-bit more for Silesia (osdb), 0.02-bit more for Silesia (reymont), 0.03-bit more for Silesia (samba), 0.03-bit more for Silesia (sao), 0.03-bit more for Silesia (webster), 0.04-bit more for Silesia (xml), and 0.03-bit more for Silesia (x-ray) relative to the entropy of the benchmark data set Silesia.

The final note for experiment 3c is that this is expected since the files in the Silesia data set are made up of very large amount of data of small integer values relative to the files in the synthetic and Wikipedia data sets. This is a favorable condition for the $\delta$-Huffman algorithm and the $\delta$-Huffman (Reconstruction with an Exception Code) algorithm has already proven in experiments 3a and 3b that the use of a special symbol, that represents the NYT and maintains a value of 0, improves on its ability to generate lower bit rate values relative to entropy.

## 5.4 Experiment 4: δ-Huffman (Update using the Sibling Property with an Exception Code)

Experiment 4 is divided into two parts: part 4a explains the compression of the synthetic data (integers) and part 4b explains the compression of real-world data from the sorted inverted index gaps from Wikipedia.

**5.4.1 Experiment 4a: Synthetic Data Set (integers).** $\delta$-Huffman (Update using the Sibling Property with an Exception Code) estimates the probability function based on the use of the formula $p_n$, the use of a special symbol that represents the NYT and maintains a value of 0, and enforcement of the sibling property. Figure 5.6 compares the $\delta$-Huffman (Update using the Sibling Property with an Exception Code) bit rate values to the entropy values of the data set

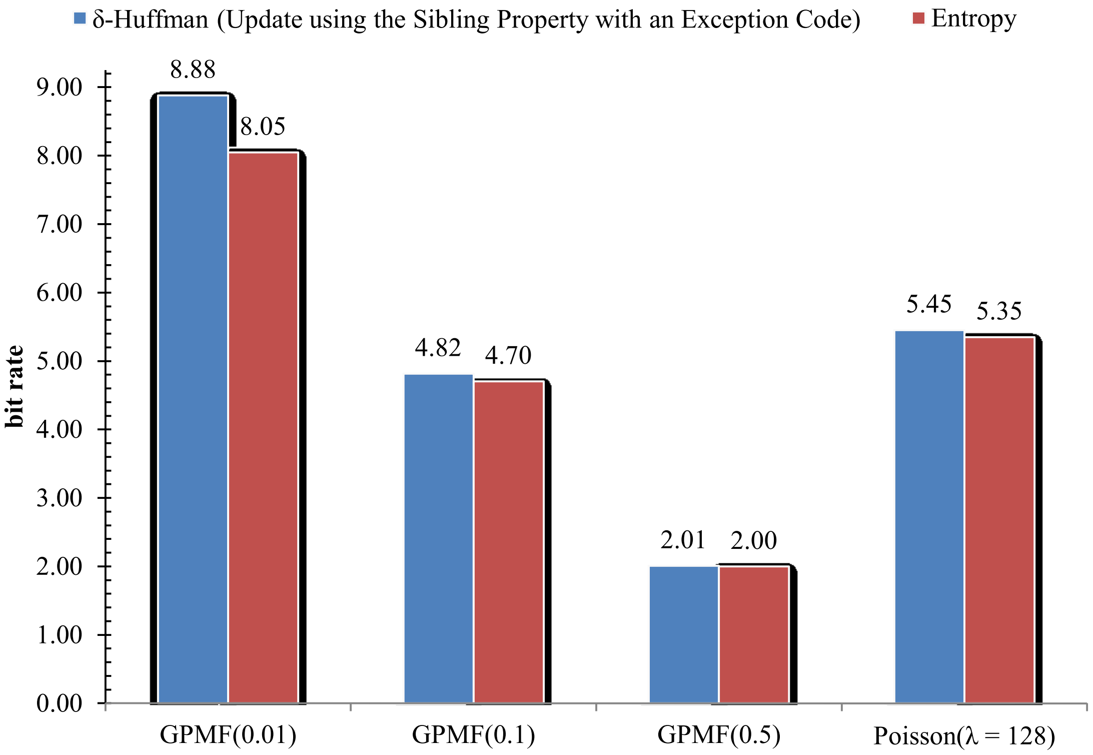


**Figure 5.6: δ-Huffman (Update using the Sibling Property with an Exception Code) bit rate values vs. Entropy**

From figure 5.6 it can be noted that, for $\delta$-Huffman (Update using the Sibling Property with an Exception Code), the bit rate values differ from their respective entropy values from a low difference of 0.01-bit for GPMF (0.5) to a high difference of 0.83-bit for GPMF (0.01).

From this experiment it can be noted that the $\delta$-Huffman (Update using the Sibling Property with an Exception Code) generates bit rate values that are higher than the entropy; 0.83-bit more for GPMF (0.01), 0.11-bit more for GPMF (0.1), 0.01-bit more for GPMF (0.5), and 0.10-bit more for Poisson (λ=128) relative to the entropy of the synthetic data set of integers.

It can be noted that the $\delta$-Huffman (Update using the Sibling Property with an Exception Code) assumptions for the estimation of the probability function cause it to:

- Generate an identical bit rate value for GPMF (0.5) and higher bit rate values; 0.05-bit more for GPMF (0.01), 0.01-bit more for GPMF (0.1), and 0.01-bit more for Poisson (λ=128) versus the bit rate values of the δ-Huffman (Reconstruction with an Exception Code).
- Generate lower bit rate values; 0.38-bit fewer for GPMF (0.01), 0.93-bit fewer for GPMF (0.1), 0.99-bit fewer for GPMF (0.5), and 0.95-bit fewer for Poisson (λ=128) versus the bit rate values of the δ-Huffman (n+1).

The final note for experiment 4a is that the $\delta$-Huffman (Update using the Sibling Property with an Exception Code) enforcement of the sibling property does not improve the bit rate value relative to entropy versus the $\delta$-Huffman (Reconstruction with an Exception Code) from experiment 3a. Due to the nature of the GPMF data as discussed in experiment 1, it appears the $\delta$-Huffman (Reconstruction with an Exception Code) variant is able to generate an estimation of the probability mass function that matches the synthetic data very closely that it left very little for the sibling property to improve on in the $\delta$-Huffman (Update using the Sibling Property with an Exception Code) variant.

**5.4.2 Experiment 4b: Real-world Data (Wikipedia).** Figure 5.7 compares the $\delta$-Huffman (Update using the Sibling Property with an Exception Code) bit rate values to the entropy values of the data set.

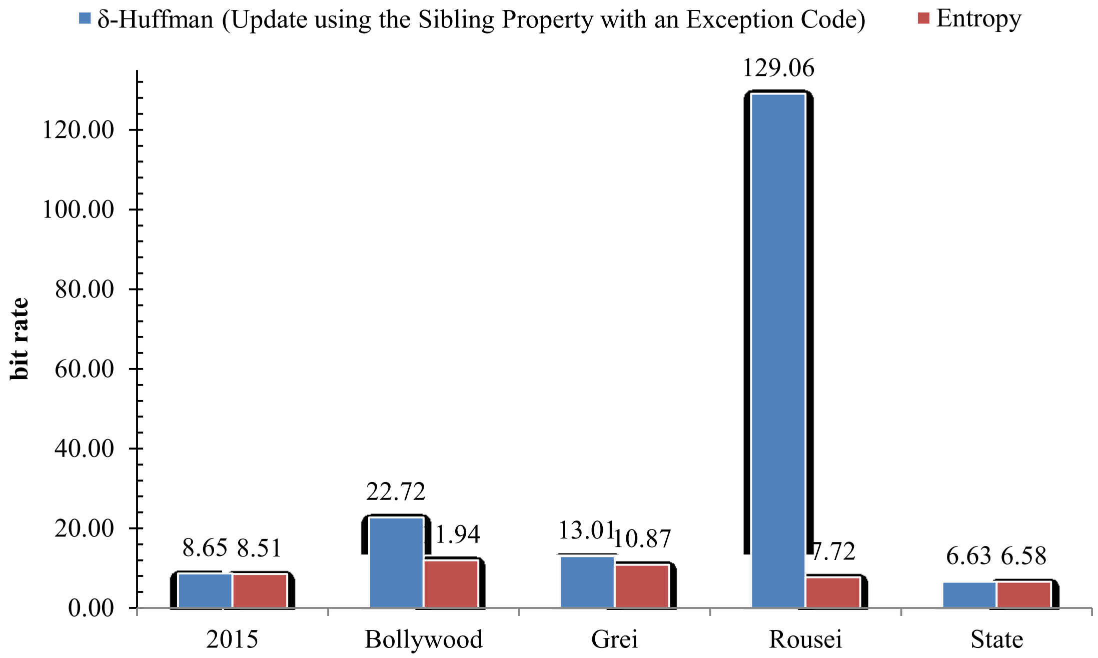


**Figure 5.7: δ-Huffman (Update using the Sibling Property with an Exception Code) bit rate values vs. Entropy**

From figure 5.7 it can be noted that, for $\delta$-Huffman (Update using the Sibling Property with an Exception Code), the bit rate values differ from their respective entropy values from a low difference of 0.04-bit for Wikipedia (State) to a high difference of 121.34 bits for Wikipedia (Rousei).

From this experiment it can be noted that the $\delta$-Huffman (Update using the Sibling Property with an Exception Code) generates a higher bit rate value relative to the entropy of the real-world data from the sorted inverted index gaps from Wikipedia.

It can be noted that the bit rate values are somewhat expected for this experiment. The $\delta$-Huffman (Update using the Sibling Property with an Exception Code) uses the formula $p_n = \frac{frequency\ count\ of\ n}{sum\ of\ all\ frequency\ counts}$, which means the values it generates are susceptible to sets with a smaller number of symbols with very large integer values like the Wikipedia (Rousei) file:

- “Rousei” is the smallest data set and contains 211 integers. The minimum integer value is 754 and maximum is 1,932,275. For this experiment it generates the largest bit rate value of 129.06 bits.

compare this to the Wikipedia (State) file:

- “State” is the biggest data set and contains 1,237,789 integers. The minimum integer value is 1 and maximum is 37,064. For this experiment it generates the smallest bit rate value of 6.63 bits.

This explains the relatively large bit rate value generated by $\delta$-Huffman (Update using the Sibling Property with an Exception Code) for the Wikipedia (Rousei) file and the relatively small bit rate value for the Wikipedia (State) file and the final experimental values seen in experiment 4b. That is, the relatively small difference of 0.04-bit more for Wikipedia (State) which has 1,237,789 index gap values and 0.15-bit more for Wikipedia (2015) which has 324,888 index gap values than for the smaller files with the relatively larger difference of 2.14 bits more for Wikipedia (Grei) which has 49,973 index gap values, 10.78 bits more for Wikipedia (Bollywood) which has 10,605 index gap values, and the relatively very large difference of 121.34 bits more for Wikipedia (Rousei) which only has 211 index gap values.

## 5.5 Experiment 5: δ-Huffman (Static Probability)

Experiment 5 is divided into two parts: part 5a explains the compression of the synthetic data (integers) and part 5b explains the compression of real-world data from the sorted inverted index gaps from Wikipedia.

**5.5.1 Experiment 5a: Synthetic Data Set (integers).** $\delta$-Huffman (Static Probability) estimates the probability function based on the use of a GPMF function with a static probability value of 0.01 if the input data is from GPMF or Wikipedia and a PMF function with a static probability value of λ=128 if the input data is from Poisson. It uses a special symbol NYT in the Huffman tree that starts with a value of 1 and decreases over time. Figure 5.8 compares the $\delta$-Huffman (Static Probability) bit rate values to the entropy values of the data set.

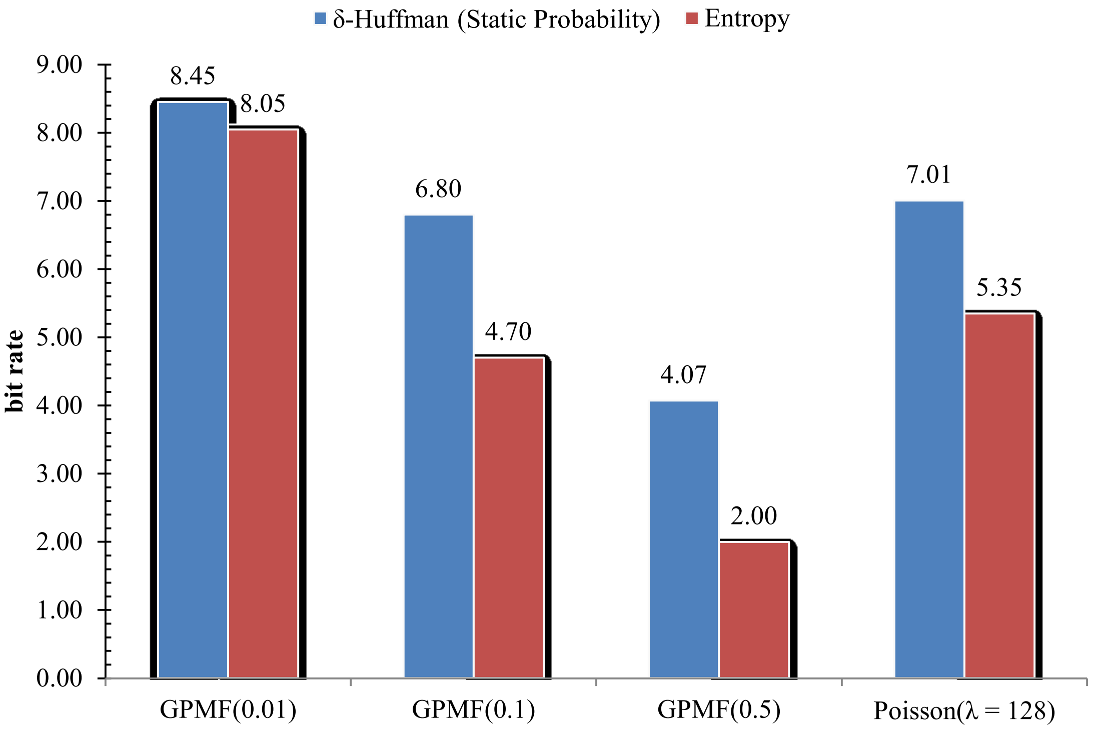


**Figure 5.8: δ-Huffman (Static Probability) bit rate values vs. Entropy**

From figure 5.8 it can be noted that, for $\delta$-Huffman (Static Probability), the bit rate values differ from their respective entropy values from a low difference of 0.41-bit for GPMF (0.01) to a high difference of 2.09 bits for GPMF (0.1).

From this experiment it can be noted that the $\delta$-Huffman (Static Probability) generates bit rate values that are higher than the entropy; 0.41-bit more for GPMF (0.01), 2.09 bits more for GPMF (0.1), 2.07 bits more for GPMF (0.5), and 1.66 bits more for Poisson (λ=128) relative to the entropy of the synthetic data set of integers.

It can be noted that the $\delta$-Huffman (Static Probability) assumptions for the estimation of the probability function cause it to:

- Generate a lower bit rate value of 0.43-bit fewer for GPMF (0.01) and higher bit rate values; 1.98 bits more for GPMF (0.1), 2.06 bits more for GPMF (0.5), and 1.55 bits more for Poisson (λ=128) versus the bit rate values of the $\delta$-Huffman (Update using the Sibling Property with an Exception Code).
- Generate a lower bit rate value of 0.38-bit fewer for GPMF (0.01) and higher bit rate values; 1.99 bits more for GPMF (0.1), 2.06 bits more for GPMF (0.5), and 1.56 bits more for Poisson (λ=128) versus the bit rate values of the $\delta$-Huffman (Reconstruction with an Exception Code).
- Generate a lower bit rate value of 0.81-bit fewer for GPMF (0.01) and higher bit rate values; 1.06 bits more for GPMF (0.1), 1.07 bits more for GPMF (0.5), and 0.60-bit more for Poisson (λ=128) versus the bit rate values of the $\delta$-Huffman (n+1).

The final note for experiment 5a is that the $\delta$-Huffman (Static Probability) bit rate values relative to entropy results for GPMF (0.01) and Poisson (λ=128) are not lower, even though it uses a GPMF function with a static probability value of 0.01 if the input data is from GPMF and a PMF function with a static probability value of λ=128 if the input data is from Poisson. This appears to be caused by the use of the special symbol NYT in the Huffman tree that starts with a value of 1 and decreases over time. This leads the $\delta$-Huffman (Static Probability) to generate worse bit rate values relative to entropy versus the $\delta$-Huffman (n+1) for the synthetic data set, with the exception of GPMF (0.01).

**5.5.2 Experiment 5b: Real-world Data (Wikipedia).** Figure 5.9 compares the $\delta$-Huffman (Static Probability) bit rate values to the entropy values of the data set.

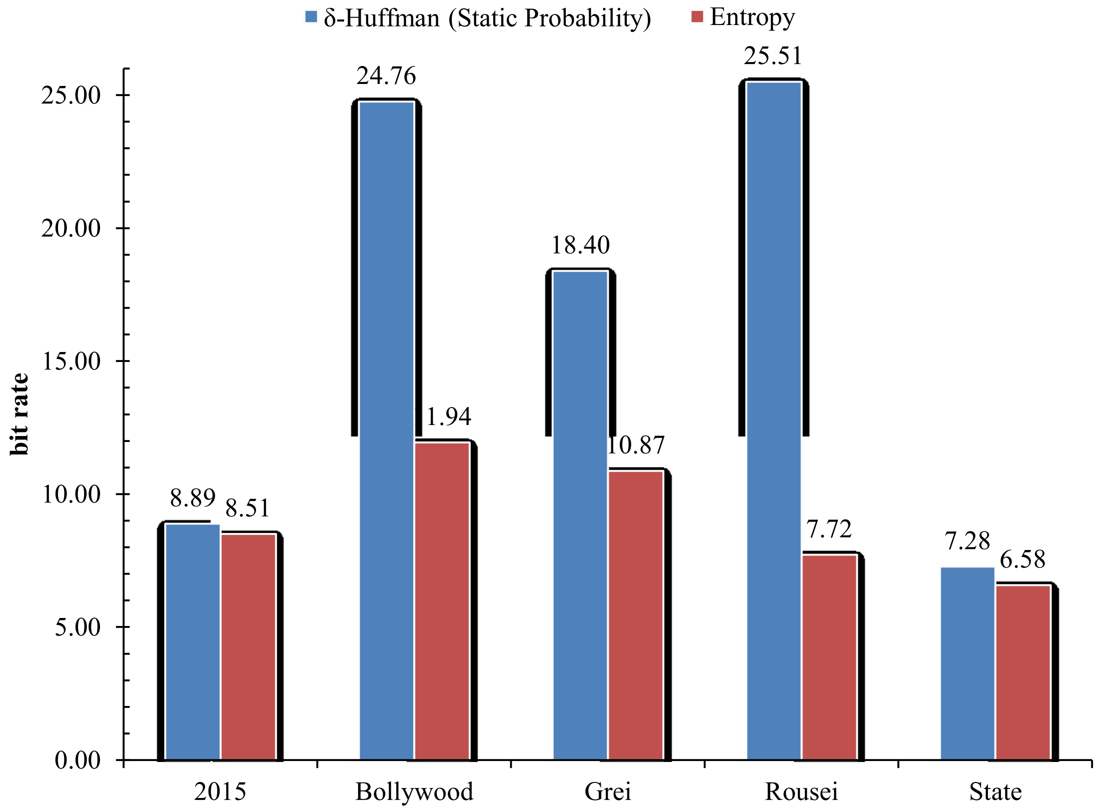


**Figure 5.9: δ-Huffman (Static Probability) bit rate values vs. Entropy**

From figure 5.9 it can be noted that, for $\delta$-Huffman (Static Probability), the bit rate values differ from their respective entropy values from a low difference of 0.38-bit for Wikipedia (2015) to a high difference of 17.79 bits for Wikipedia (Rousei).

From this experiment it can be noted that the $\delta$-Huffman (Static Probability) generates higher bit rate values relative to the entropy of the real-world data from the sorted inverted index gaps from Wikipedia.

As in experiment 4b, it can be noted the δ-Huffman (Static Probability) for larger files generates the relatively small difference of 0.38-bit more for Wikipedia (2015) and 0.69-bit more for Wikipedia (State) than for the smaller files with the relatively larger difference of 7.52 bits more for Wikipedia (Grei), 12.82 bits more for Wikipedia (Bollywood), and 17.79 bits more for Wikipedia (Rousei).

It can be noted that δ-Huffman (Static Probability) generates a lower bit rate value of 103.55 bits for Wikipedia (Rousei) and higher bit rate values; 0.23-bit more for Wikipedia (2015), 2.04 bits more for Wikipedia (Bollywood), 5.38 bits more for Wikipedia (Grei), and 0.65-bit more for Wikipedia (State) versus the bit rate values of the δ-Huffman (Update using the Sibling Property with an Exception Code).

The final note for experiment 5b is that unlike experiment 5a, this experiment shows the $\delta$-Huffman (Static Probability) use of a GPMF with a static value of 0.01 and use of a special symbol NYT in the Huffman tree that starts with a value of 1 makes it less susceptible to the number of symbols seen in a data set file. This allows the $\delta$-Huffman (Static Probability) to generate better bit rate values relative to entropy versus the δ-Huffman (Update using the Sibling Property with an Exception Code) from experiment 4b.

## 5.6 Experiment 6: δ-Huffman (Sibling/Static Probability)

Experiment 6 is divided into two parts: part 6a explains the compression of the synthetic data (integers) and part 6b explains the compression of real-world data from the sorted inverted index gaps from Wikipedia.

**5.6.1 Experiment 6a: Synthetic Data Set (integers).** $\delta$-Huffman (Sibling/Static Probability) estimates the probability function based on the use of a GPMF function with a static probability value of 0.01 for input data from GPMF or Wikipedia and a PMF function with a static probability value of λ=128 for input data from Poisson. It uses a special symbol NYT in the Huffman tree that starts with a value of 1 and decreases over time and enforces the sibling property. Figure 5.10 compares the $\delta$-Huffman (Sibling/Static Probability) bit rate values to the entropy values of the data set.

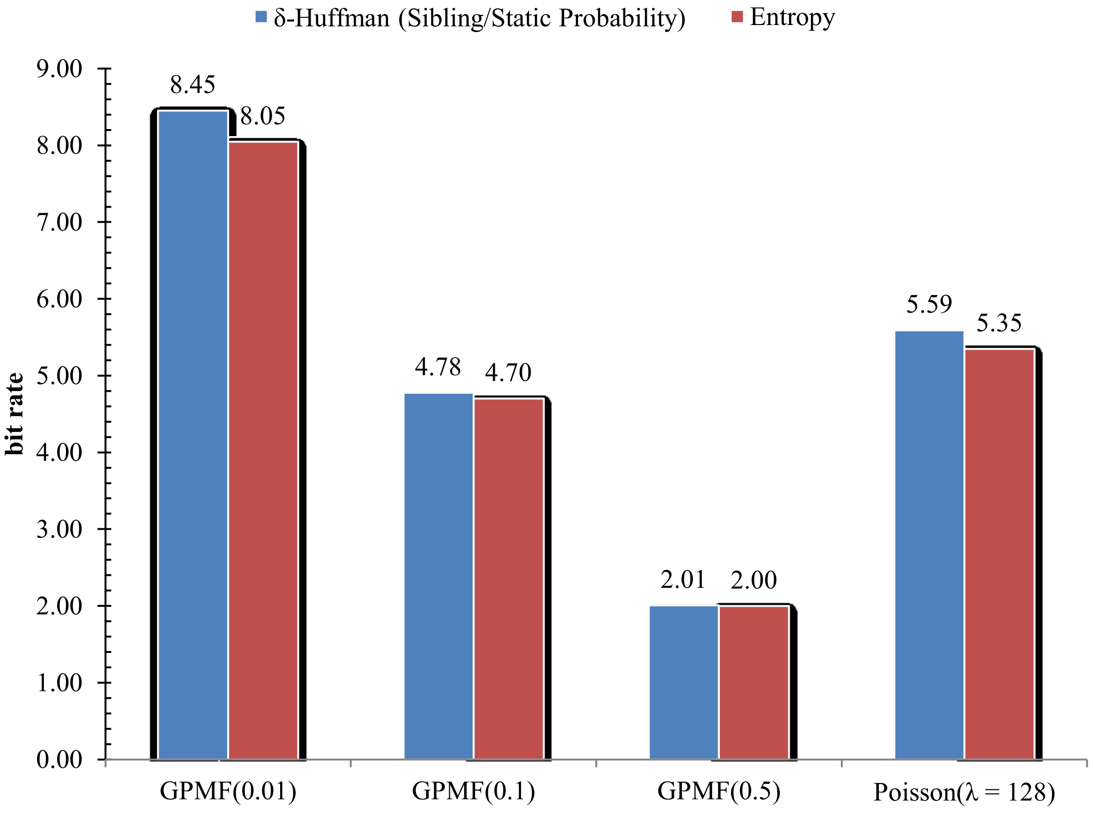


**Figure 5.10: δ-Huffman (Sibling/Static Probability) bit rate values vs. Entropy**

From figure 5.10 it can be noted that, for $\delta$-Huffman (Sibling/Static Probability), the bit rate values differ from their respective entropy values from a low difference of 0.01-bit for GPMF (0.5) to a high difference of 0.41-bit for GPMF (0.01).

From this experiment it can be noted that the $\delta$-Huffman (Sibling/Static Probability) generates bit rate values that are higher than the entropy; 0.41-bit more for GPMF (0.01), 0.08-bit more for GPMF (0.1), 0.01-bit more for GPMF (0.5), and 0.24-bit more for Poisson (λ=128) relative to the entropy of the synthetic data set of integers.

It can be noted that the $\delta$-Huffman (Sibling/Static Probability) assumptions for the estimation of the probability function cause it to:

- Generate lower bit rate values; 2.02 bits fewer for GPMF (0.1), 2.06 bits fewer for GPMF (0.5), and 1.41 bits fewer for Poisson (λ=128) and generates an identical bit rate value for GPMF (0.01) versus the bit rate values of the $\delta$-Huffman (Static Probability).
- Generate lower bit rate values; 0.43-bit fewer for GPMF (0.01) and 0.04-bit fewer for GPMF (0.1). It generates an identical bit rate value for GPMF (0.5) and a higher bit rate value of 0.14-bit more for Poisson (λ=128) versus the bit rate values of the $\delta$-Huffman (Update using Sibling Property with an Exception Code).
- Generate lower bit rate values; 0.37-bit fewer for GPMF (0.01) and 0.03-bit fewer for GPMF (0.1). It generates an identical bit rate for GPMF (0.5) and a higher bit rate value of 0.15-bit more for Poisson (λ=128) versus the bit rate values of the $\delta$-Huffman (Reconstruction with an Exception Code).
- Generate lower bit rate values; 0.80-bit fewer for GPMF (0.01), 0.96-bit fewer for GPMF (0.1), 0.99-bit fewer for GPMF (0.5), and 0.81-bit fewer for Poisson (λ=128) versus the bit rate values of the $\delta$-Huffman (n+1).

The final note for experiment 6a is that the $\delta$-Huffman (Sibling/Static Probability) additional enforcement of the sibling property allow it to generate better bit rate values relative to entropy for GPMF (0.5), GPMF (0.1), and Poisson versus the $\delta$-Huffman (Static Probability) from experiment 5a even though both use the same GPMF formula with a static 0.01 probability value and a NYT that starts at 1 and decreases over time. This is in contrast to the relatively small bit rate values relative to entropy differences made between the $\delta$-Huffman (Update using the Sibling Property with an Exception

Code) from experiment 4a and the $\delta$-Huffman (Reconstruction with an Exception Code) from experiment 3a and their use of a special symbol NYT that has a permanent value of 0. The $\delta$-Huffman (Sibling/Static Probability) generates better bit rate values relative to entropy versus the $\delta$-Huffman (n+1).

**5.6.2 Experiment 6b: Real-world Data (Wikipedia).** Figure 5.11 compares the $\delta$-Huffman (Sibling/Static Probability) bit rate values to the entropy values of the data set.

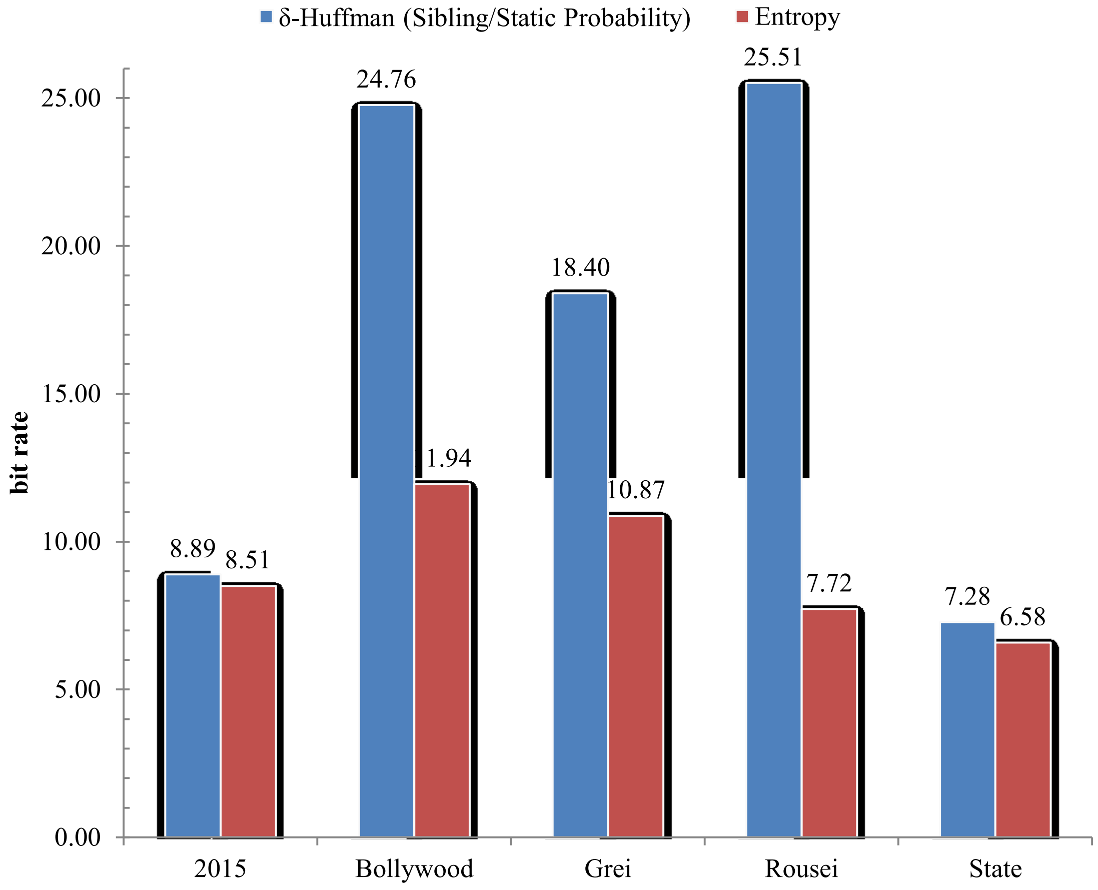


**Figure 5.11: δ-Huffman (Sibling/Static Probability) bit rate values vs. Entropy**

From figure 5.11 it can be noted that, for $\delta$-Huffman (Sibling/Static Probability), the bit rate values differ from their respective entropy values from a low difference of 0.38-bit for Wikipedia (2015) to a high difference of 17.79 bits for Wikipedia (Rousei).

From this experiment it can be noted that the $\delta$-Huffman (Sibling/Static Probability) generates higher bit rate values relative to the entropy of the real-world data from the sorted inverted index gaps from Wikipedia.

As in experiments 4b and 5b, it can be noted that the δ-Huffman (Sibling/Static Probability) for larger files generates the relatively small difference of 0.38-bit more for Wikipedia (2015) and 0.69-bit more for Wikipedia (State) than for the smaller files with the relatively larger difference of 7.52 bits more for Wikipedia (Grei), 12.82 bits more for Wikipedia (Bollywood), and 17.79 bits more for Wikipedia (Rousei).

It can be noted that the $\delta$-Huffman (Sibling/Static Probability) assumptions for the estimation of the probability function cause it to:

- Generate identical bit rate values for Wikipedia (2015), Wikipedia (Bollywood), Wikipedia (Grei), Wikipedia (Rousei), and Wikipedia (State) versus the bit rate values of the $\delta$-Huffman (Static Probability).
- Generate a lower bit rate value of 103.55 bits for Wikipedia (Rousei) and higher bit rate values; 0.23-bit more for Wikipedia (2015), 2.04 bits more for Wikipedia (Bollywood), 5.38 bits more for Wikipedia (Grei), and 0.65-bit more for Wikipedia (State) versus the bit rate values of the $\delta$-Huffman (Update using the Sibling Property with an Exception Code).

It can be noted that the $\delta$-Huffman (Sibling/Static Probability) additional enforcement of the sibling property does not improve on any bit rate values versus the $\delta$-Huffman (Static Probability) like it did in experiment 6a. This is somewhat similar to how the $\delta$-Huffman (Update using the Sibling Property with an Exception Code) additional enforcement of the sibling property from experiment 4a hardly improved the bit rate value results relative to entropy of the $\delta$-Huffman (Reconstruction with an Exception Code) from experiment 3a for the synthetic data.

The final note for experiment 6b is much like how the $\delta$-Huffman (Reconstruction with an Exception Code) was able to generate an estimation of the probability mass function that matched the synthetic data very closely that it left very little for the sibling property to improve on in the $\delta$-Huffman (Update using the Sibling Property with an Exception Code) variant, the $\delta$-Huffman (Static Probability) estimation of the probability mass function matches the Wikipedia data very closely that the $\delta$-Huffman (Sibling/Static Probability) additional enforcement of the sibling property has nothing to improve on.

## 5.7 Experiment 7: δ-Huffman (Dynamic Probability P2)

Experiment 7 is divided into three parts: part 7a explains the compression of the synthetic data (integers), part 7b explains the compression of real-world data from the sorted inverted index gaps from Wikipedia, and part 7c explains the compression of the benchmark data set Silesia.

**5.7.1 Experiment 7a: Synthetic Data Set (integers).** $\delta$-Huffman (Dynamic Probability P2) estimates the probability function based on the use of a special symbol that represents the NYT and maintains a value of 0 and the formula P2. Figure 5.12 compares the $\delta$-Huffman (Dynamic Probability P2) bit rate values to the entropy values of the data set.

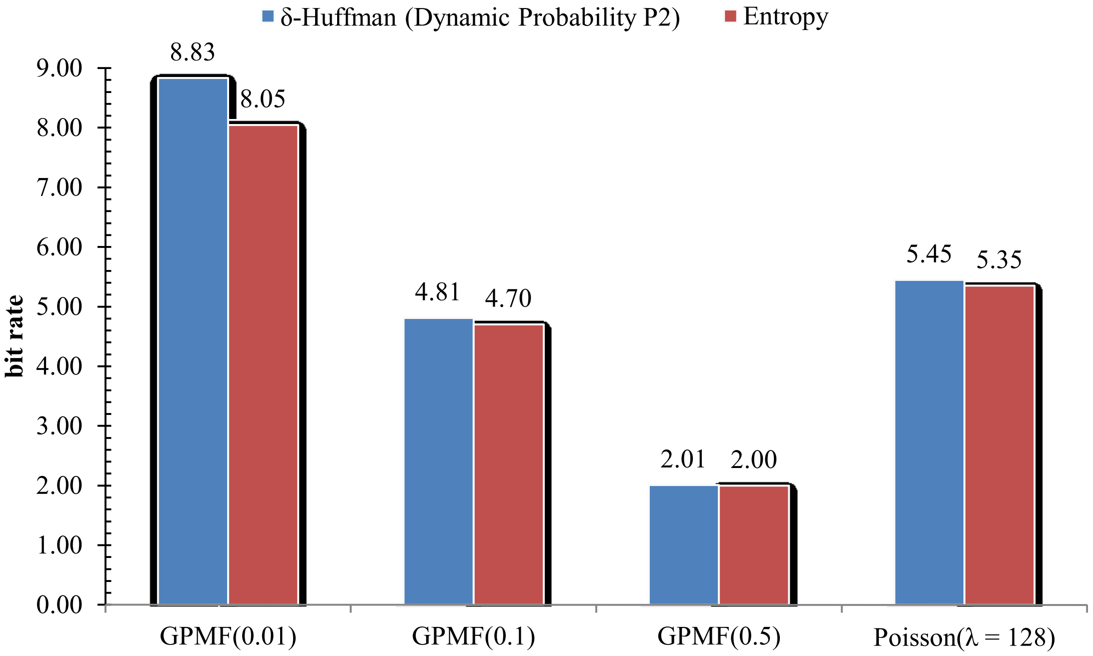


**Figure 5.12: δ-Huffman (Dynamic Probability P2) bit rate values vs. Entropy**

From figure 5.12 it can be noted that, for $\delta$-Huffman (Dynamic Probability P2), the bit rate values differ from their respective entropy values from a low difference of 0.01-bit for GPMF (0.5) to a high difference of 0.78-bit for GPMF (0.01).

From this experiment it can be noted that the $\delta$-Huffman (Dynamic Probability P2) generates bit rate values that are higher than the entropy; 0.78-bit more for GPMF (0.01), 0.11-bit more for GPMF (0.1), 0.01-bit more for GPMF (0.5), and 0.10-bit more for Poisson (λ=128) relative to the entropy of the synthetic data set of integers.

It can be noted that the $\delta$-Huffman (Dynamic Probability P2) assumptions for the estimation of the probability function cause it to:

- Generate a lower bit rate value of 0.15-bit fewer for Poisson (λ=128). It generates an identical bit rate value for GPMF (0.5) and higher bit rate values; 0.38-bit more for GPMF (0.01) and 0.03-bit more for GPMF (0.1) versus the bit rate values of the $\delta$-Huffman (Sibling/Static Probability).

- Generate lower bit rate values; 1.99 bits fewer for GPMF (0.1), 2.06 bits fewer for GPMF (0.5), and 1.56 bits fewer for Poisson (λ=128) and a higher bit rate value of 0.38-bit more for GPMF (0.01) versus the bit rate values of the $\delta$-Huffman (Static Probability).
- Generate lower bit rate values; 0.05-bit fewer for GPMF (0.01), 0.01-bit fewer for GPMF (0.1), and 0.01-bit fewer for Poisson (λ=128) and an identical bit rate value for GPMF (0.5) versus the bit rate values of the $\delta$-Huffman (Update using Sibling Property with an Exception Code).
- Generate identical bit rate values for GPMF (0.01), GPMF (0.1), GPMF (0.5), and Poisson (λ=128) versus the bit rate values of the $\delta$-Huffman (Reconstruction with an Exception Code).
- Generate lower bit rate values; 0.43-bit fewer for GPMF (0.01), 0.93-bit fewer for GPMF (0.1), 0.99-bit fewer for GPMF (0.5), and 0.96-bit fewer for Poisson (λ=128) versus the bit rate values of the $\delta$-Huffman (n+1).

It can be noted that the $\delta$-Huffman (Dynamic Probability P2) and $\delta$-Huffman (Reconstruction with an Exception Code) are both similar in that they both use a special symbol NYT that starts with and maintains a value of 0. The main difference is that the $\delta$-Huffman (Dynamic Probability P2) uses the formula P2 which assumes N is the sum of all input values while the $\delta$-Huffman (Reconstruction with an Exception Code) uses the formula $p_n$ which assumes N is the total number of symbols that have been seen so far. This difference appears to be offset by the nature of the GPMF data as discussed in experiment 1, small integers with multiple repetitions. This causes the $\delta$-Huffman

(Dynamic Probability P2) to generate bit rate values relative to entropy that are similar to the $\delta$-Huffman (Reconstruction with an Exception Code).

Note that since the $\delta$-Huffman (Dynamic Probability P2) performs the same as the $\delta$-Huffman (Reconstruction with an Exception Code), it is expected that it  generates better bit rate values relative to entropy than the $\delta$-Huffman (Update using Sibling Property with an Exception Code) variant which the $\delta$-Huffman (Reconstruction with an Exception Code) variant performed better than in experiment 4a and the $\delta$-Huffman (n+1) variant which the $\delta$-Huffman (Reconstruction with an Exception Code) variant performed better than in experiment 3a.

Note that the $\delta$-Huffman (Dynamic Probability P2) variant is different than the $\delta$-Huffman (Static Probability) variant in two main areas, the $\delta$-Huffman (Dynamic Probability P2) variant uses an NYT symbol that starts with and maintains a value of 0 and the formula P2 for the estimation of the probability function while the $\delta$-Huffman (Static Probability) variant uses an NYT symbol that starts with a value of 1 and decreases over time and the GPMF function with a static probability value of 0.01 for the estimation of the probability function. This allows the $\delta$-Huffman (Dynamic Probability P2) to generate better bit rate values relative to entropy for GPMF (0.5), GPMF (0.1), and Poisson while the $\delta$-Huffman (Static Probability) only generates a better bit rate value relative to entropy for GPMF (0.01) for the synthetic data set.

The final note for experiment 7a is that the $\delta$-Huffman (Sibling/Static Probability) inclusion of the enforcement of the sibling property allows it to generate better bit rate values relative to entropy for GPMF (0.01) and GPMF (0.1) and a similar bit rate value relative to entropy for GPMF (0.5) versus the $\delta$-Huffman (Dynamic Probability P2). The

$\delta$-Huffman (Dynamic Probability P2) only generates a better bit rate value relative to entropy for Poisson in this experiment. A plausible explanation for this is the $\delta$-Huffman (Sibling/Static Probability) use of a GPMF function with a static probability value of 0.01 for the estimation of the probability function.

**5.7.2 Experiment 7b: Real-world Data (Wikipedia).** Figure 5.13 compares the $\delta$-Huffman (Dynamic Probability P2) bit rate values to the entropy values of the data set.

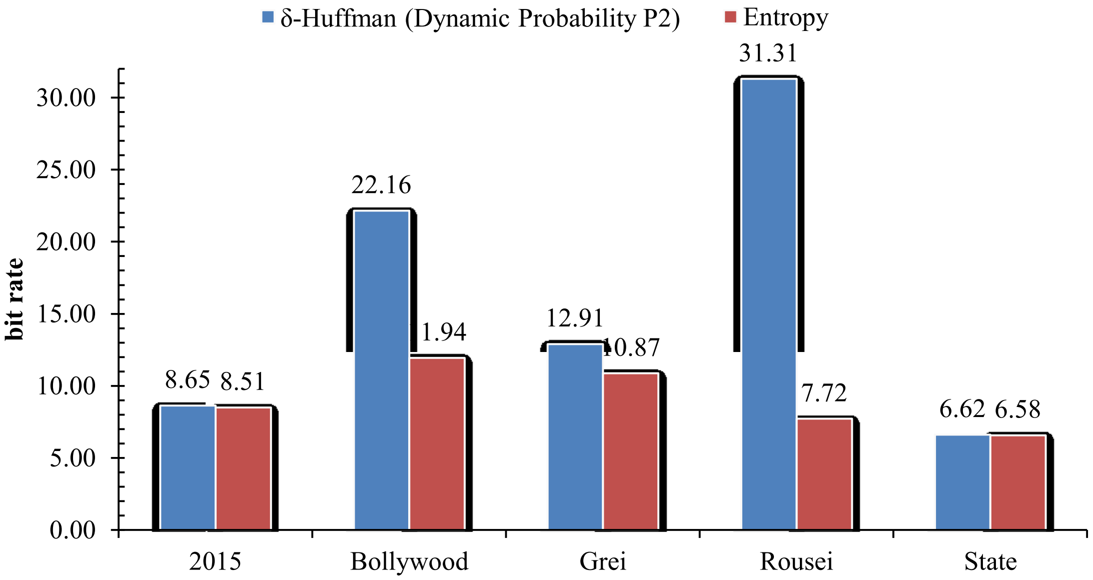


**Figure 5.13: δ-Huffman (Dynamic Probability P2) bit rate values vs. Entropy**

From figure 5.13 it can be noted that, for $\delta$-Huffman (Dynamic Probability P2), the bit rate values differ from their respective entropy values from a low difference of 0.04-bit for Wikipedia (State) to a high difference of 23.59 bits for Wikipedia (Rousei).

From this experiment it can be noted that the $\delta$-Huffman (Dynamic Probability P2) generates higher bit rate values relative to the entropy of the real-world data from the sorted inverted index gaps from Wikipedia. As in experiments 4b, 5b, and 6b it can be noted the δ-Huffman (Dynamic Probability P2) for larger files generates the relatively small difference of 0.14-bit more for Wikipedia (2015) and 0.04-bit more for Wikipedia

(State) than for the smaller files with the relatively larger difference of 2.04 bits more for Wikipedia (Grei), 10.22 bits more for Wikipedia (Bollywood), and 23.59 bits more for Wikipedia (Rousei).

It can be noted that the $\delta$-Huffman (Dynamic Probability P2) assumptions for the estimation of the probability function cause it to:

- Generate lower bit rate values; 0.24-bit fewer for Wikipedia (2015), 2.60 bits fewer for Wikipedia (Bollywood), 5.48 bits fewer for Wikipedia (Grei), and 0.65-bit fewer for Wikipedia (State) and a higher bit rate value of 5.80 bits more for Wikipedia (Rousei) versus the bit rate values of the $\delta$-Huffman (Sibling/Static Probability).
- Generate lower bit rate values; 0.24-bit fewer for Wikipedia (2015), 2.60 bits fewer for Wikipedia (Bollywood), 5.48 bits fewer for Wikipedia (Grei), and 0.65-bit fewer for Wikipedia (State) and a higher bit rate value of 5.80 bits more for Wikipedia (Rousei) versus the bit rate values of the $\delta$-Huffman (Static Probability).
- Generate lower bit rate values; 0.01-bit fewer for Wikipedia (2015), 0.56-bit fewer for Wikipedia (Bollywood), 0.10-bit fewer for Wikipedia (Grei), and 97.75 bits fewer for Wikipedia (Rousei) and an identical bit rate value for Wikipedia (State) versus the bit rate values of the $\delta$-Huffman (Update using the Sibling Property with an Exception Code).

It can be noted that the $\delta$-Huffman (Dynamic Probability P2) and $\delta$-Huffman (Update using the Sibling Property with an Exception Code) are similar in that they both use a special symbol NYT that starts with and maintains a value of 0. The main difference is that $\delta$-Huffman (Dynamic Probability P2) uses the formula P2 while $\delta$-Huffman (Update using the Sibling Property with an Exception Code) uses the formula $p_n$. This allows the $\delta$-Huffman (Dynamic Probability P2) to generate a similar bit rate value as $\delta$-Huffman (Update using the Sibling Property with an Exception Code) for Wikipedia (2015) and slightly better bit rate values relative to entropy for the other Wikipedia files.

The final note for experiment 7b is that the $\delta$-Huffman (Dynamic Probability P2) use of a special symbol NYT that starts with and maintains a value of 0 and the formula P2 while the $\delta$-Huffman (Sibling/Static Probability) uses a NYT symbol that starts with a value of 1 and decreases over time and the GPMF function with a static probability value of 0.01 for the estimation of the probability function, leads the $\delta$-Huffman (Dynamic Probability P2) to generate better bit rate values relative to entropy for all Wikipedia files except for Wikipedia (Rousei). For similar reasons, the $\delta$-Huffman (Dynamic Probability P2) generates better bit rate values relative to entropy for all Wikipedia files, except for Wikipedia (Rousei), versus the $\delta$-Huffman (Static Probability) variant.

**5.7.3 Experiment 7c: Benchmark Data Set (Silesia).** Figure 5.14 compares the $\delta$-Huffman (Dynamic Probability P2) bit rate values to the entropy values of the data set.

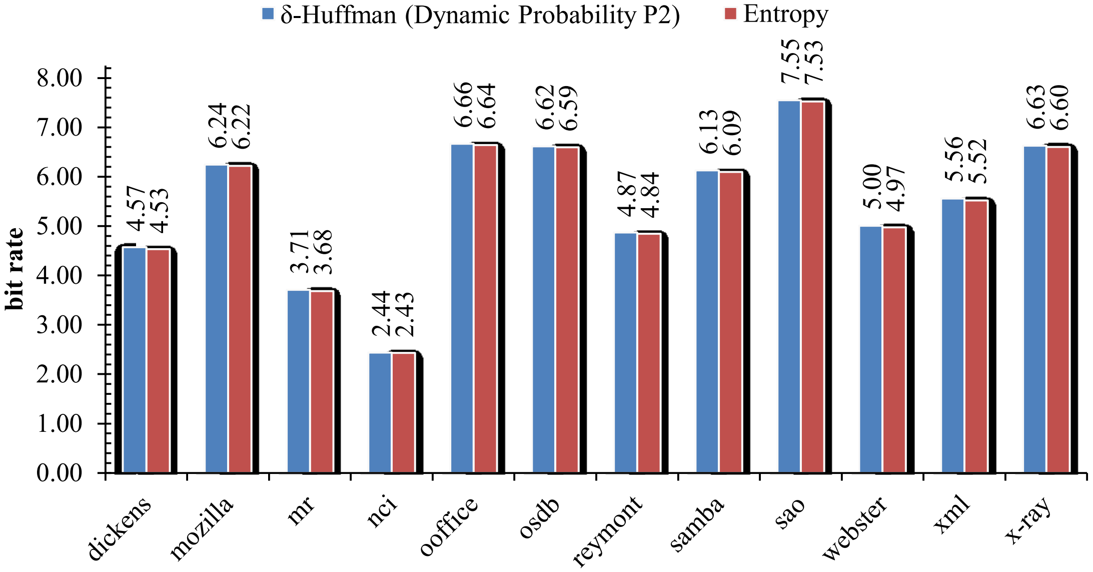


**Figure 5.14: δ-Huffman (Dynamic Probability P2) bit rate values vs. Entropy**

From figure 5.14 it can be noted that, for $\delta$-Huffman (Dynamic Probability P2), the bit rate values differ from their respective entropy values from a low difference of 0.01-bit for Silesia (nci) to a high difference of 0.04-bit for Silesia (dickens) and Silesia (xml).

From this experiment it can be noted that the $\delta$-Huffman (Dynamic Probability P2) generates bit rate values that are higher than the entropy; 0.04-bit more for Silesia (dickens), 0.02-bit more for Silesia (mozilla), 0.03-bit more for Silesia (mr), 0.01-bit more for Silesia (nci), 0.02-bit more for Silesia (ooffice), 0.02-bit more for Silesia (osdb), 0.02-bit more for Silesia (reymont), 0.03-bit more for Silesia (samba), 0.03-bit more for Silesia (sao), 0.03-bit more for Silesia (webster), 0.04-bit more for Silesia (xml), and 0.03-bit more for Silesia (x-ray) relative to the entropy of the benchmark data set Silesia.

It can be noted that the $\delta$-Huffman (Dynamic Probability P2) generates essentially identical bit rate values for; Silesia (dickens), Silesia (mozilla), Silesia (mr), Silesia (nci), Silesia (ooffice), Silesia (osdb), Silesia (reymont), Silesia (samba), Silesia (sao), Silesia (webster), Silesia (xml), and Silesia (x-ray) versus the bit rate values of the $\delta$-Huffman (Reconstruction with an Exception Code).

The final note for experiment 7c is that the $\delta$-Huffman (Dynamic Probability P2) and the $\delta$-Huffman (Reconstruction with an Exception Code) are similar in that they both use a special symbol NYT that starts with and maintains a value of 0. The main difference is that the $\delta$-Huffman (Dynamic Probability P2) uses the formula P2 while the $\delta$-Huffman (Reconstruction with an Exception Code) uses the formula $p_n$. However, due to the nature of the Silesia data files as discussed in experiment 3c, the $\delta$-Huffman (Reconstruction with an Exception Code) was able to generate an estimation of the probability mass function that matched the Silesia data very closely that it left very little for the $\delta$-Huffman (Dynamic Probability P2) variant to improve on.

## 5.8 Experiment 8: δ-Huffman (Dynamic Probability P1/Slice)

Experiment 8 is divided into three parts: part 8a explains the compression of the synthetic data (integers), part 8b explains the compression of real-world data from the sorted inverted index gaps from Wikipedia, and part 8c explains the compression of the benchmark data set Silesia.

**5.8.1 Experiment 8a: Synthetic Data Set (integers).** $\delta$-Huffman (Dynamic Probability P1/Slice) estimates the probability function based on the use of a special symbol that represents the NYT and starts with a value of 1 and decreases over time, use of the formula P1, and segmentation of the input data into slices $(L)$. Figure 5.15 compares the $\delta$-Huffman (Dynamic Probability P1/Slice) bit rate values to the entropy values of the data set.

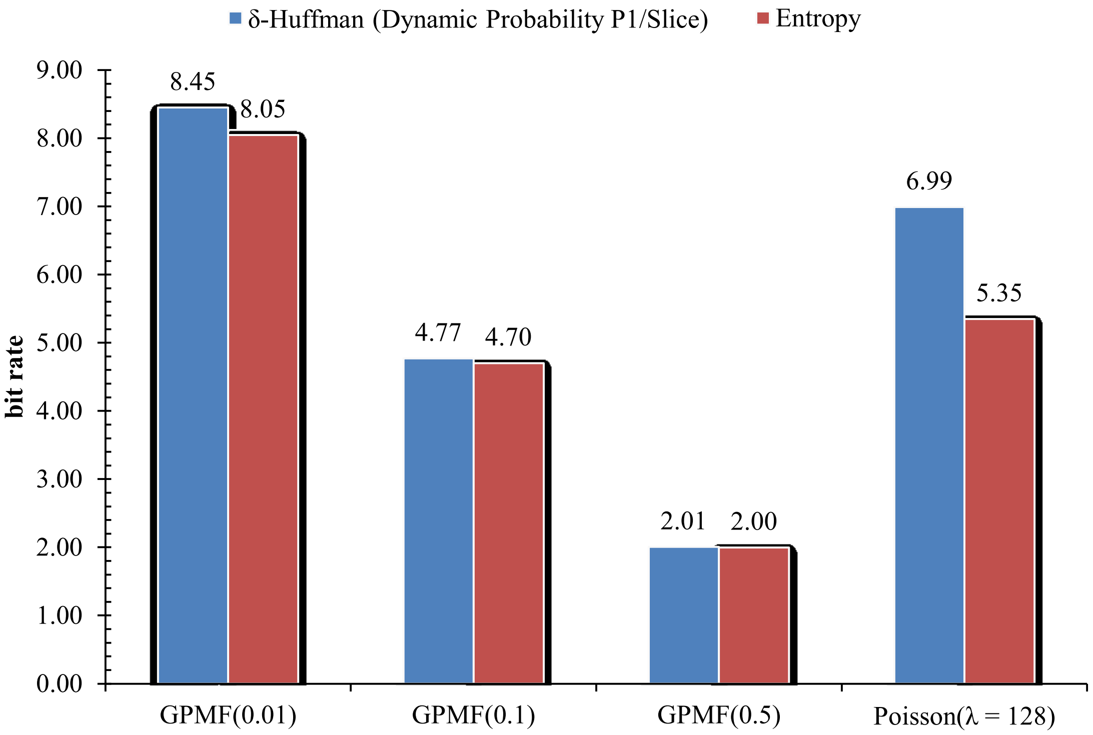


**Figure 5.15: δ-Huffman (Dynamic Probability P1/Slice) bit rate values vs. Entropy**

From figure 5.15 it can be noted that, for $\delta$-Huffman (Dynamic Probability P1/Slice), the bit rate values differ from their respective entropy values from a low difference of 0.01-bit for GPMF (0.5) to a high difference of 1.64 bits for Poisson (λ = 128).

From this experiment it can be noted that the $\delta$-Huffman (Dynamic Probability P1/Slice) generates bit rate values that are higher than the entropy; 0.40-bit more for GPMF (0.01), 0.07-bit more for GPMF (0.1), 0.01-bit more for GPMF (0.5), and 1.64 bits more for Poisson (λ=128) relative to the entropy of the synthetic data set of integers.

It can be noted that the $\delta$-Huffman (Dynamic Probability P1/Slice) assumptions for the estimation of the probability function cause it to:

- Generate lower bit rate values; 0.38-bit fewer for GPMF (0.01) and 0.04-bit fewer for GPMF (0.1), an identical bit rate value for GPMF (0.5), and a higher bit rate value of 1.55 bits more for Poisson (λ = 128) versus the bit rate values of the $\delta$-Huffman (Dynamic Probability P2).

- Generate a lower bit rate value of 0.01-bit fewer for GPMF (0.1), identical bit rate values for GPMF (0.01) and GPMF (0.5), and a higher bit rate value of 1.40 bits more for Poisson (λ = 128) versus the bit rate values of the $\delta$-Huffman (Sibling/Static Probability).
- Generate lower bit rate values; 2.03 bits fewer for GPMF (0.1), 2.06 bits fewer for GPMF (0.5), and 0.01-bit fewer for Poisson (λ = 128) and an identical bit rate value for GPMF (0.01) versus the bit rate values of the $\delta$-Huffman (Static Probability).
- Generate lower bit rate values; 0.43-bit fewer for GPMF (0.01) and 0.05-bit fewer for GPMF (0.1), an identical bit rate value for GPMF (0.5), and a higher bit rate value of 1.54 bits more for Poisson (λ = 128) versus the bit rate values of the $\delta$-Huffman (Update using Sibling Property with an Exception Code).
- Generate lower bit rate values; 0.38-bit fewer for GPMF (0.01) and 0.04-bit fewer for GPMF (0.1), an identical bit rate value for GPMF (0.5), and a higher bit rate value of 1.55 bits more for Poisson (λ = 128) versus the bit rate values of the $\delta$-Huffman (Reconstruction with an Exception Code).
- Generate lower bit rate values; 0.81-bit fewer for GPMF (0.01), 0.97-bit fewer for GPMF (0.1), and 1.00-bit fewer for GPMF (0.5) and a higher bit rate value of 0.59-bit more for Poisson (λ = 128) versus the bit rate values of the $\delta$-Huffman (n+1).

It can be noted that the $\delta$-Huffman (Dynamic Probability P1/Slice) and $\delta$-Huffman (Sibling/Static Probability) are similar in that they both use an NYT that starts with a value of 1 and decreases over time. The main difference is that the $\delta$-Huffman (Dynamic Probability P1/Slice) uses the formula P1 and segments the input data into slices while the $\delta$-Huffman (Sibling/Static Probability) uses a GPMF with a static probability of 0.01. This leads the $\delta$-Huffman (Dynamic Probability P1/Slice) to generate similar bit rate values relative to entropy for GPMF (0.5) and GPMF (0.01) and a slightly better bit rate value relative to entropy for GPMF (0.1) while the $\delta$-Huffman (Sibling/Static Probability) generates a better bit rate value relative to entropy for Poisson.

Note that the $\delta$-Huffman (Static Probability) is similar to the $\delta$-Huffman (Sibling/Static Probability) with the enforcement of the sibling property removed. Therefore, it is expected the $\delta$-Huffman (Static Probability) generates worse bit rate values relative to entropy versus the $\delta$-Huffman (Dynamic Probability P1/Slice). The exception is a similar bit rate value relative to entropy for GPMF (0.01). A plausible explanation for this is the $\delta$-Huffman (Static Probability) use of a GPMF with a static probability of 0.01 for the estimation of the probability function while the $\delta$-Huffman (Dynamic Probability P1/Slice) uses the formula P1 for the estimation of the probability function.

Note that the $\delta$-Huffman (Dynamic Probability P1/Slice) uses the formula P1, uses a special symbol that represents the NYT and starts with a value of 1 and decreases over time, and segments the input data into slices. The $\delta$-Huffman (Dynamic Probability P2) uses the formula P2 and a special symbol that represents the NYT that starts with and maintains a value of 0. This leads the $\delta$-Huffman (Dynamic Probability P1/Slice) to generate better bit rate values relative to entropy for GPMF (0.1) and GPMF (0.01), a bit rate value relative to entropy that is similar to the Huffman (Dynamic Probability P2) for GPMF (0.5) and a worse bit rate value relative to entropy for Poisson. A plausible explanation for this is the $\delta$-Huffman (Dynamic Probability P1/Slice) use of the formula P1 while the $\delta$-Huffman (Dynamic Probability P2) uses the formula P2.

Note that the $\delta$-Huffman (Dynamic Probability P1/Slice) and $\delta$-Huffman (Update using Sibling Property with an Exception Code) main difference is that the $\delta$-Huffman (Dynamic Probability P1/Slice) uses the formula P1, uses a special symbol that represents the NYT and starts with a value of 1 and decreases over time, and segments the input data into slices while the $\delta$-Huffman (Update using Sibling Property with an Exception Code) uses the formula $p_n$. This leads the $\delta$-Huffman (Dynamic Probability P1/Slice) to generate better bit rate values relative to entropy for GPMF (0.1) and GPMF (0.01), a bit rate value relative to entropy that is similar to the $\delta$-Huffman (Update using Sibling Property with an Exception Code) for GPMF (0.5) and a worse bit rate value relative to entropy for Poisson.

Note that the $\delta$-Huffman (Dynamic Probability P1/Slice) has nothing in common with the $\delta$-Huffman (Reconstruction with an Exception Code). The $\delta$-Huffman (Reconstruction with an Exception Code) is similar to the $\delta$-Huffman (Update using Sibling Property with an Exception Code) in that they both use the formula $p_n$. Therefore, the similar bit rate values relative to entropy performance of the $\delta$-Huffman (Dynamic Probability P1/Slice) versus the $\delta$-Huffman (Reconstruction with an Exception Code) is expected.

The final note for experiment 8a is that the $\delta$-Huffman (Dynamic Probability P1/Slice) generates better bit rate values relative to entropy versus $\delta$-Huffman (n+1) for all GPMF files, but worse for the Poisson file. A plausible explanation for these results is the $\delta$-Huffman (Dynamic Probability P1/Slice) use of the formula P1 for its estimation of the probability function.

**5.8.2 Experiment 8b: Real-world Data (Wikipedia).** Figure 5.16 compares the $\delta$-Huffman (Dynamic Probability P1/Slice) bit rate values to the entropy values of the data set.

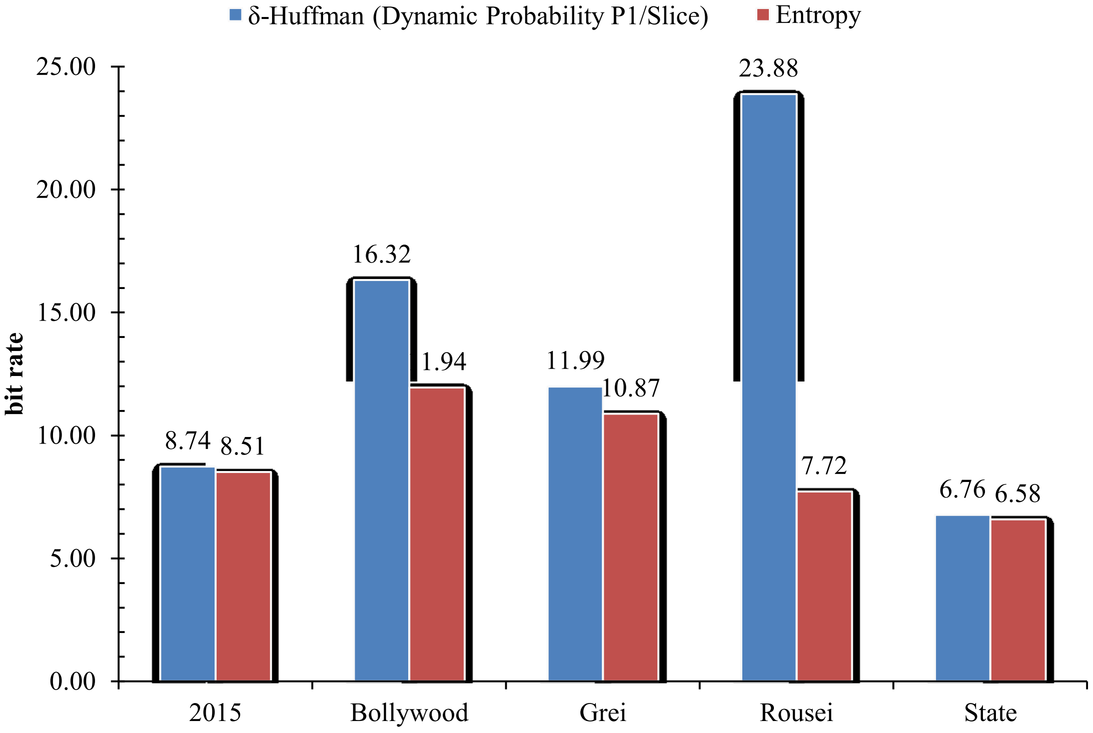


**Figure 5.16: δ-Huffman (Dynamic Probability P1/Slice) bit rate values vs. Entropy**

From figure 5.16 it can be noted that, for $\delta$-Huffman (Dynamic Probability P1/Slice), the bit rate values differ from their respective entropy values from a low difference of 0.18-bit for Wikipedia (State) to a high difference of 16.16 bits for Wikipedia (Rousei).

From this experiment it can be noted that the $\delta$-Huffman (Dynamic Probability P1/Slice) generates a higher bit rate value relative to the entropy of the real-world data from the sorted inverted index gaps from Wikipedia. As in experiments 4b, 5b, 6b, and 7b it can be noted the δ-Huffman (Dynamic Probability P1/Slice) for larger files generates the relatively small difference of 0.23-bit more for Wikipedia (2015) and 0.18-bit more for Wikipedia (State) than for the smaller files with the relatively larger difference of 1.12 bits more for Wikipedia (Grei), 4.37 bits more for Wikipedia (Bollywood), and 16.16 bits more for Wikipedia (Rousei).

It can be noted that the $\delta$-Huffman (Dynamic Probability P1/Slice) assumptions for the estimation of the probability function cause it to:

- Generate lower bit rate values; 5.85 bits fewer for Wikipedia (Bollywood), 0.92-bit fewer for Wikipedia (Grei), and 7.43 bits fewer for Wikipedia (Rousei) and higher bit rate values; 0.09-bit more for Wikipedia (2015) and 0.14-bit more for Wikipedia (State) versus the bit rate values of the $\delta$-Huffman (Dynamic Probability P2).
- Generate lower bit rate values; 0.15-bit fewer for Wikipedia (2015), 8.45 bits fewer for Wikipedia (Bollywood), 6.40 bits fewer for Wikipedia (Grei), 1.63 bits fewer for Wikipedia (Rousei), and 0.51-bit fewer for Wikipedia (State) versus the bit rate values of the $\delta$-Huffman (Sibling/Static Probability).
- Generate lower bit rate values; 0.15-bit fewer for Wikipedia (2015), 8.45 bits fewer for Wikipedia (Bollywood), 6.40 bits fewer for Wikipedia (Grei), 1.63 bits fewer for Wikipedia (Rousei), and 0.51-bit fewer for Wikipedia (State) versus the bit rate values of the $\delta$-Huffman (Static Probability).

- Generate lower bit rate values; 6.41 bits fewer for Wikipedia (Bollywood), 1.02 bits fewer for Wikipedia (Grei), and 105.18 bits fewer for Wikipedia (Rousei) and higher bit rate values; 0.08-bit more for Wikipedia (2015) and 0.14-bit more for Wikipedia (State) versus the bit rate values of the $\delta$-Huffman (Update using the Sibling Property with an Exception Code).

It can be noted that the $\delta$-Huffman (Dynamic Probability P1/Slice) uses the formula P1, uses a special symbol that represents the NYT and starts with a value of 1 and decreases over time, and segments the input data into slices. The $\delta$-Huffman (Dynamic Probability P2) uses the formula P2 and a special symbol that represents the NYT that starts with and maintains a value of 0. This leads the $\delta$-Huffman (Dynamic Probability P1/Slice) to generate better bit rate values relative to entropy for the smaller size files Wikipedia (Grei), Wikipedia (Bollywood), and Wikipedia (Rousei) while the $\delta$-Huffman (Dynamic Probability P2) generates better bit rate values relative to entropy for the larger size files Wikipedia (2015) and Wikipedia (State).

Note that the $\delta$-Huffman (Static Probability) and the $\delta$-Huffman (Sibling/Static Probability) both generate similar bit rate values relative to entropy, and the $\delta$-Huffman (Dynamic Probability P1/Slice) generates better bit rate values relative to entropy than both of them. The main difference is the $\delta$-Huffman (Dynamic Probability P1/Slice) uses the formula P1 and segments the input data into slices while the $\delta$-Huffman (Sibling/Static Probability) and $\delta$-Huffman (Static Probability) use a GPMF with a static probability of 0.01 for the estimation of the probability function. This leads the $\delta$-Huffman (Dynamic Probability P1/Slice) to generate better bit rate values relative to entropy for all Wikipedia files versus the $\delta$-Huffman (Sibling/Static Probability) and $\delta$-Huffman (Static Probability).

Note that the $\delta$-Huffman (Dynamic Probability P1/Slice) generates better bit rate values relative to entropy for the smaller size files Wikipedia (Grei), Wikipedia (Bollywood), and Wikipedia (Rousei) than for the larger size files Wikipedia (2015) and Wikipedia (State).

The final note for experiment 8b is that the $\delta$-Huffman (Dynamic Probability P1/Slice) and $\delta$-Huffman (Update using Sibling Property with an Exception Code) main difference is that the $\delta$-Huffman (Dynamic Probability P1/Slice) uses the formula P1, a special symbol that represents the NYT and starts with a value of 1 and decreases over time, and segments the input data into slices while the $\delta$-Huffman (Update using Sibling Property with an Exception Code) uses the formula $p_n$. This leads the $\delta$-Huffman (Dynamic Probability P1/Slice) to generate better bit rate values relative to entropy for the smaller size files Wikipedia (Grei), Wikipedia (Bollywood), and Wikipedia (Rousei) while the $\delta$-Huffman (Update using the Sibling Property with an Exception Code) generates better bit rate values relative to entropy for the larger size files Wikipedia (2015) and Wikipedia (State).

**5.8.3 Experiment 8c: Benchmark Data Set (Silesia).** Figure 5.17 compares the $\delta$-Huffman (Dynamic Probability P1/Slice) bit rate values to the entropy values of the data set.

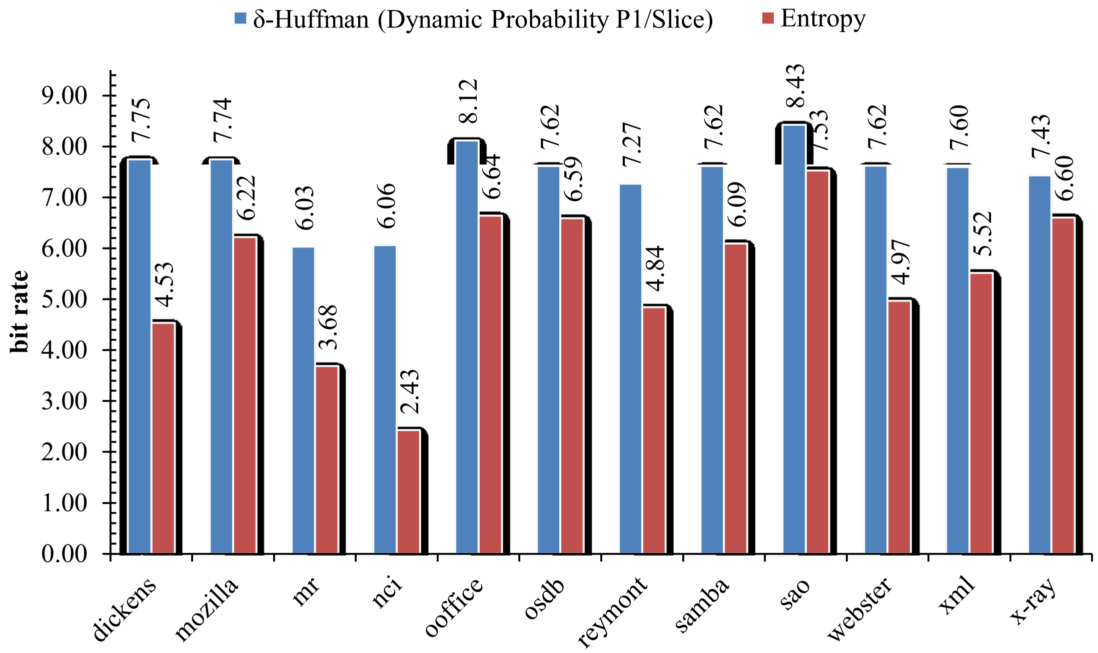


**Figure 5.17: δ-Huffman (Dynamic Probability P1/Slice) bit rate values vs. Entropy**

From figure 5.17 it can be noted that, for $\delta$-Huffman (Dynamic Probability P1/Slice), the bit rate values differ from their respective entropy values from a low difference of 0.83-bit for Silesia (x-ray) to a high difference of 3.63 bits for Silesia (nci).

From this experiment it can be noted that the $\delta$-Huffman (Dynamic Probability P1/Slice) generates bit rate values that are higher than the entropy; 3.22 bits more for Silesia (dickens), 1.52 bits more for Silesia (mozilla), 2.35 bits more for Silesia (mr), 3.63 bits more for Silesia (nci), 1.48 bits more for Silesia (ooffice), 1.02 bits more for Silesia (osdb), 2.42 bits more for Silesia (reymont), 1.52 bits more for Silesia (samba), 0.90-bit more for Silesia (sao), 2.65 bits more for Silesia (webster), 2.08 bits more for Silesia (xml), and 0.83-bit more for Silesia (x-ray) relative to the entropy of the benchmark data set Silesia.

It can be noted that the $\delta$-Huffman (Dynamic Probability P1/Slice) assumptions for the estimation of the probability function cause it to:

- Generate higher bit rate values; 3.18 bits more for Silesia (dickens), 1.50 bits more for Silesia (mozilla), 2.32 bits more for Silesia (mr), 3.62 bits more for Silesia (nci), 1.45 bits more for Silesia (ooffice), 1.00-bit more for Silesia (osdb), 2.40 bits more for Silesia (reymont), 1.49 bits more for Silesia (samba), 0.88-bit more for Silesia (sao), 2.62 bits more for Silesia (webster), 2.04 bits more for Silesia (xml), and 0.80-bit more for Silesia (x-ray) versus the bit rate values of the $\delta$-Huffman (Dynamic Probability P2).
- Generate higher bit rate values; 3.18 bits more for Silesia (dickens), 1.50 bits more for Silesia (mozilla), 2.32 bits more for Silesia (mr), 3.62 bits more for Silesia (nci), 1.45 bits more for Silesia (ooffice), 1.00-bit more for Silesia (osdb), 2.40 bits more for Silesia (reymont), 1.49 bits more for Silesia (samba), 0.88-bit more for Silesia (sao), 2.62 bits more for Silesia (webster), 2.04 bits more for Silesia (xml), and 0.80-bit more for Silesia (x-ray) versus the bit rate values of the $\delta$-Huffman (Reconstruction with an Exception Code).

The final note for experiment 8c is that the $\delta$-Huffman (Dynamic Probability P1/Slice) use of the formula P1, a NYT that starts with a value of 1 and decreases over time, and use of the slice method leads the $\delta$-Huffman (Dynamic Probability P1/Slice) to generate worse bit rate values relative to entropy versus the $\delta$-Huffman (Dynamic Probability P2) and $\delta$-Huffman (Reconstruction with an Exception Code) for the Silesia data set.

## 5.9 Experiment 9: δ-Huffman (Dynamic Probability P2/Slice)

Experiment 9 is divided into three parts: part 9a explains the compression of the synthetic data (integers), part 9b explains the compression of real-world data from the sorted inverted index gaps from Wikipedia, and part 9c explains the compression of the benchmark data set Silesia.

**5.9.1 Experiment 9a: Synthetic Data Set (integers).** $\delta$-Huffman (Dynamic Probability P2/Slice) estimates the probability function based on use of a special symbol that represents the NYT and maintains a value of 0, use of the formula P2, and segmentation of the input data into slices $(L)$. Figure 5.18 compares the $\delta$-Huffman (Dynamic Probability P2/Slice) bit rate values to the entropy values of the data set.

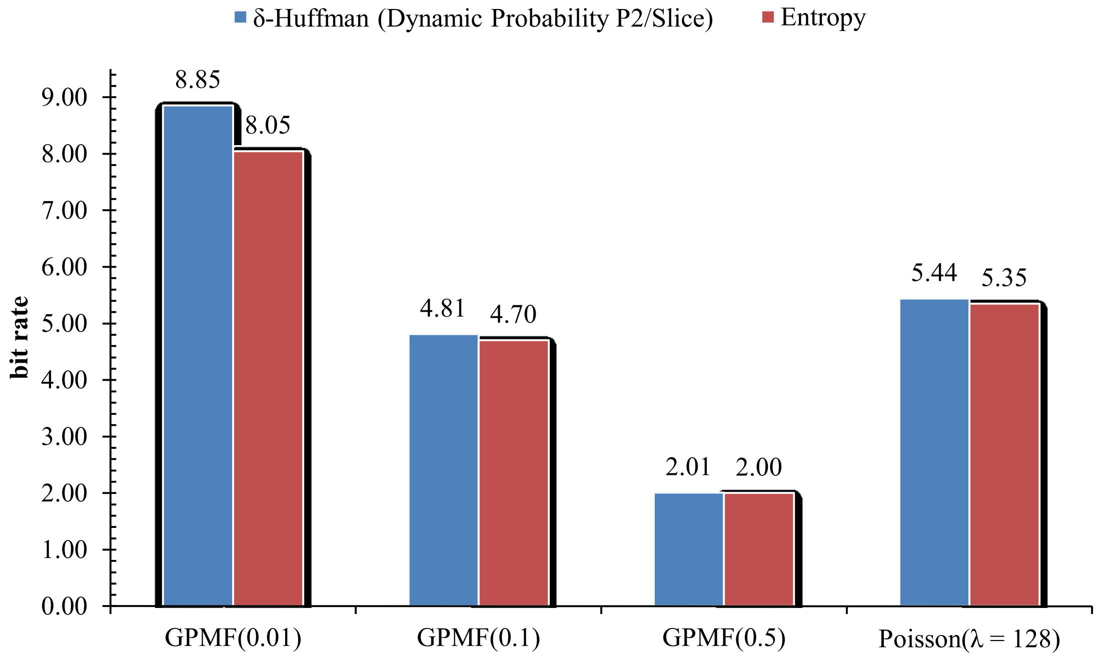


**Figure 5.18: δ-Huffman (Dynamic Probability P2/Slice) bit rate values vs. Entropy**

From figure 5.18 it can be noted that, for $\delta$-Huffman (Dynamic Probability P2/Slice), the bit rate values differ from their respective entropy values from a low difference of 0.01-bit for GPMF (0.5) to a high difference of 0.81-bit for GPMF (0.01).

From this experiment it can be noted that the $\delta$-Huffman (Dynamic Probability P2/Slice) generates bit rate values that are higher than the entropy; 0.81-bit more for GPMF (0.01), 0.11-bit more for GPMF (0.1), 0.01-bit more for GPMF (0.5), and 0.09-bit more for Poisson (λ=128) relative to the entropy of the synthetic data set of integers.

It can be noted that the $\delta$-Huffman (Dynamic Probability P2/Slice) assumptions for the estimation of the probability function cause it to:

- Generate a lower bit rate value of 1.55 bits fewer for Poisson (λ = 128), an identical bit rate value for GPMF (0.5), and higher bit rate values; 0.41-bit more for GPMF (0.01) and 0.04-bit more for GPMF (0.1) versus the bit rate values of the $\delta$-Huffman (Dynamic Probability P1/Slice).
- Generate a lower bit rate value of 0.01-bit fewer for Poisson (λ = 128), identical bit rate values for GPMF (0.1) and GPMF (0.5), and a higher bit rate value of 0.02-bit more for GPMF (0.01) versus the bit rate values of the $\delta$-Huffman (Dynamic Probability P2).
- Generate a lower bit rate value of 0.15-bit fewer for Poisson (λ = 128), an identical bit rate value for GPMF (0.5), and higher bit rate values; 0.40-bit more for GPMF (0.01) and 0.03-bit more for GPMF (0.1) versus the bit rate values of the $\delta$-Huffman (Sibling/Static Probability).
- Generate lower bit rate values; 1.99 bits fewer for GPMF (0.1), 2.06 bits fewer for GPMF (0.5), and 1.57 bits fewer for Poisson (λ = 128) and a higher bit rate value of 0.40-bit more for GPMF (0.01) versus the bit rate values of the $\delta$-Huffman (Static Probability).

- Generate lower bit rate values; 0.03-bit fewer for GPMF (0.01), 0.01-bit fewer for GPMF (0.1), and 0.01-bit fewer for Poisson (λ = 128) and an identical bit rate value for GPMF (0.5) versus the bit rate values of the $\delta$-Huffman (Update using Sibling Property with an Exception Code).
- Generate a lower bit rate value of 0.01-bit fewer for Poisson (λ = 128), identical bit rate values for GPMF (0.1) and GPMF (0.5), and a higher bit rate value of 0.03-bit more for GPMF (0.01) versus the bit rate values of the $\delta$-Huffman (Reconstruction with an Exception Code).
- Generate lower bit rate values; 0.40-bit fewer for GPMF (0.01), 0.93-bit fewer for GPMF (0.1), 0.99-bit fewer for GPMF (0.5), and 0.96-bit fewer for Poisson (λ = 128) versus the bit rate values of the $\delta$-Huffman (n+1).

The final note for experiment 9a is that the $\delta$-Huffman (Dynamic Probability P2/Slice) additional use of the slice method cause it to generate a better bit rate value relative to entropy for Poisson, similar bit rate values relative to entropy for GPMF (0.1) and GPMF (0.5), and a worse bit rate value relative to entropy for GPMF (0.01) versus the $\delta$-Huffman (Dynamic Probability P2).

**5.9.2 Experiment 9b: Real-world Data (Wikipedia).** Figure 5.19 compares the $\delta$-Huffman (Dynamic Probability P2/Slice) bit rate values to the entropy values of the data set.

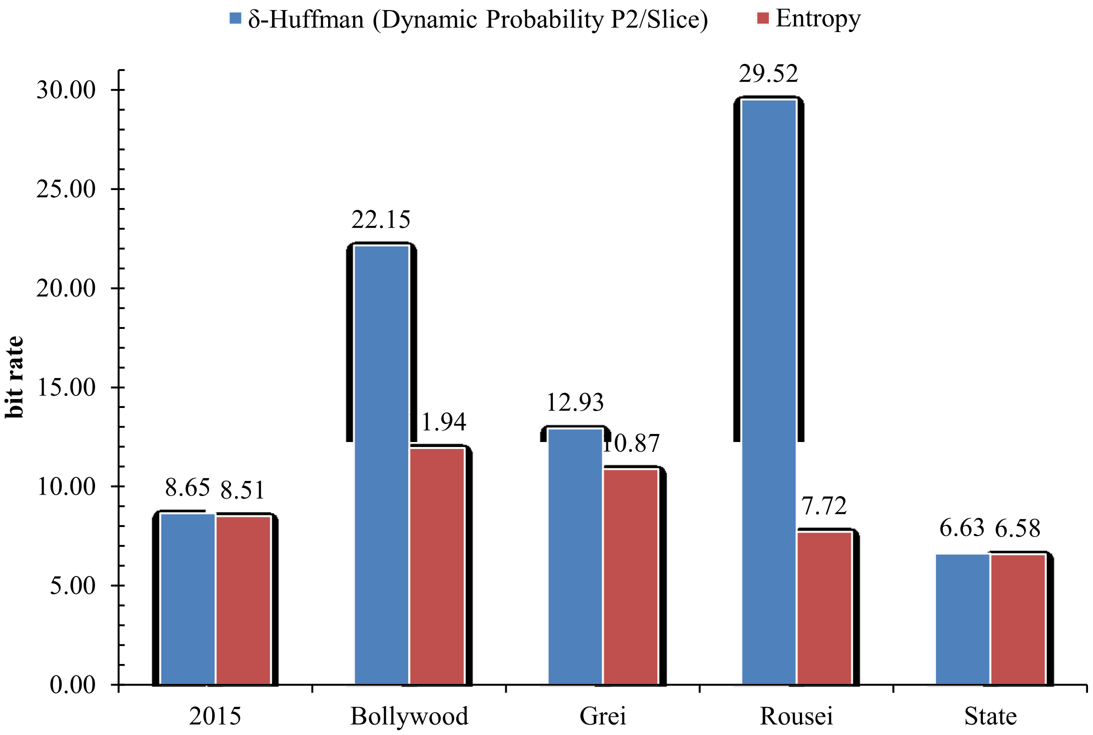


**Figure 5.19: δ-Huffman (Dynamic Probability P2/Slice) bit rate values vs. Entropy**

From figure 5.19 it can be noted that, for $\delta$-Huffman (Dynamic Probability P2/Slice), the bit rate values differ from their respective entropy values from a low difference of 0.04-bit for Wikipedia (State) to a high difference of 21.80 bits for Wikipedia (Rousei).

From this experiment it can be noted that the $\delta$-Huffman (Dynamic Probability P2/Slice) generates higher bit rate values relative to the entropy of the real-world data from the sorted inverted index gaps from Wikipedia. As in experiments 4b, 5b, 6b, 7b, and 8b it can be noted the δ-Huffman (Dynamic Probability P2/Slice) for larger files generates the relatively small difference of 0.14-bit more for Wikipedia (2015) and 0.04-bit more for Wikipedia (State) than for the smaller files with the relatively larger difference of 2.06 bits more for Wikipedia (Grei), 10.21 bits more for Wikipedia (Bollywood), and 21.80 bits more for Wikipedia (Rousei).

It can be noted that the $\delta$-Huffman (Dynamic Probability P2/Slice) assumptions for the estimation of the probability function cause it to:

- Generate lower bit rate values; 0.09-bit fewer for Wikipedia (2015) and 0.14-bit fewer for Wikipedia (State) and higher bit rate values; 5.84 bits more for Wikipedia (Bollywood), 0.94-bit more for Wikipedia (Grei), and 5.64 bits more for Wikipedia (Rousei) versus the bit rate values of the $\delta$-Huffman (Dynamic Probability P1/Slice).
- Generate lower bit rate values; 0.01-bit fewer for Wikipedia (Bollywood) and 1.79 bits fewer for Wikipedia (Rousei), identical bit rate values for Wikipedia (2015) and Wikipedia (State), and a higher bit rate value of 0.02-bit more for Wikipedia (Grei) versus the bit rate values of the $\delta$-Huffman (Dynamic Probability P2).
- Generate lower bit rate values; 0.24-bit fewer for Wikipedia (2015), 2.61 bits fewer for Wikipedia (Bollywood), 5.46 bits fewer for Wikipedia (Grei), and 0.65-bit fewer for Wikipedia (State) and a higher bit rate value of 4.00 bits more for Wikipedia (Rousei) versus the bit rate values of the $\delta$-Huffman (Sibling/Static Probability).
- Generate lower bit rate values; 0.24-bit fewer for Wikipedia (2015), 2.61 bits fewer for Wikipedia (Bollywood), 5.46 bits fewer for Wikipedia (Grei), and 0.65-bit fewer for Wikipedia (State) and a higher bit rate value of 4.00 bits more for Wikipedia (Rousei) versus the bit rate values of the $\delta$-Huffman (Static Probability).

- Generate lower bit rate values; 0.57-bit fewer for Wikipedia (Bollywood), 0.08-bit fewer for Wikipedia (Grei), and 99.55 bits fewer for Wikipedia (Rousei) and identical bit rate values for Wikipedia (2015) and Wikipedia (State) versus the bit rate values of the $\delta$-Huffman (Update using the Sibling Property with an Exception Code).

It can be noted that the $\delta$-Huffman (Dynamic Probability P2/Slice) and the $\delta$-Huffman (Dynamic Probability P1/Slice) main difference is the $\delta$-Huffman (Dynamic Probability P2/Slice) use of the formula P2 and a NYT that starts with and maintains a value of 0 while the $\delta$-Huffman (Dynamic Probability P1/Slice) uses the formula P1 and a NYT that starts with a value of 1 and decreases over time. This leads the $\delta$-Huffman (Dynamic Probability P2/Slice) to generate better bit rate values relative to entropy for the larger size files Wikipedia (2015) and Wikipedia (State) while the $\delta$-Huffman (Dynamic Probability P1/Slice) generates better bit rate values relative to entropy for the smaller size files Wikipedia (Grei), Wikipedia (Bollywood), and Wikipedia (Rousei).

The final note for experiment 9b is that the addition of the slice method leads the $\delta$-Huffman (Dynamic Probability P2/Slice) to generate better bit rate values relative to entropy for Wikipedia (Bollywood) and Wikipedia (Rousei), a similar bit rate value relative to entropy for Wikipedia (2015) and worse bit rate values relative to entropy for Wikipedia (State) and Wikipedia (Grei) versus the $\delta$-Huffman (Dynamic Probability P2).

**5.9.3 Experiment 9c: Benchmark Data Set (Silesia).** Figure 5.20 compares the $\delta$-Huffman (Dynamic Probability P2/Slice) bit rate values to the entropy values of the data set.

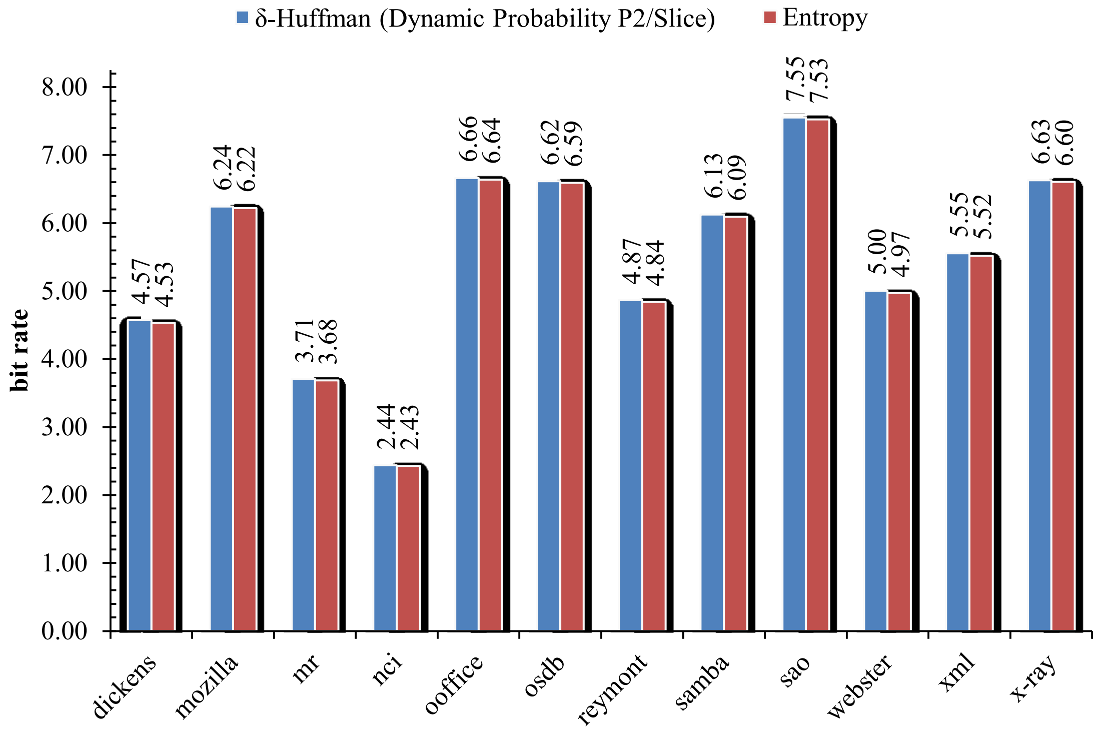


**Figure 5.20: δ-Huffman (Dynamic Probability P2/Slice) bit rate values vs. Entropy**

From figure 5.20 it can be noted that, for $\delta$-Huffman (Dynamic Probability P2/Slice), the bit rate values differ from their respective entropy values from a low difference of 0.01-bit for Silesia (nci) to a high difference of 0.04-bit for Silesia (dickens) and Silesia (xml).

From this experiment it can be noted that the $\delta$-Huffman (Dynamic Probability P2/Slice) generates bit rate values that are higher than the entropy; 0.04-bit more for Silesia (dickens), 0.02-bit more for Silesia (mozilla), 0.03-bit more for Silesia (mr), 0.01-bit more for Silesia (nci), 0.03-bit more for Silesia (ooffice), 0.02-bit more for Silesia (osdb), 0.03-bit more for Silesia (reymont), 0.03-bit more for Silesia (samba), 0.03-bit more for Silesia (sao), 0.03-bit more for Silesia (webster), 0.04-bit more for Silesia (xml), and 0.03-bit more for Silesia (x-ray) relative to the entropy of the benchmark data set Silesia.

It can be noted that the $\delta$-Huffman (Dynamic Probability P2/Slice) assumptions for the estimation of the probability function cause it to:

- Generate lower bit rate values; 3.18 bits fewer for Silesia (dickens), 1.50 bits fewer for Silesia (mozilla), 2.32 bits fewer for Silesia (mr), 3.62 bits fewer for Silesia (nci), 1.45 bits fewer for Silesia (ooffice), 1.00-bit fewer for Silesia (osdb), 2.40 bits fewer for Silesia (reymont), 1.49 bits fewer for Silesia (samba), 0.88-bit fewer for Silesia (sao), 2.62 bits fewer for Silesia (webster), 2.04 bits fewer for Silesia (xml), and 0.80-bit fewer for Silesia (x-ray) versus the bit rate values of the $\delta$-Huffman (Dynamic Probability P1/Slice).
- Generate a lower bit rate value; 0.01-bit fewer for Silesia (xml) and identical bit rate values for; Silesia (dickens), Silesia (mozilla), Silesia (mr), Silesia (nci), Silesia (ooffice), Silesia (osdb), Silesia (reymont), Silesia (samba), Silesia (sao), Silesia (webster), and Silesia (x-ray) versus the bit rate values of the $\delta$-Huffman (Dynamic Probability P2).
- Generate identical bit rate values for; Silesia (dickens), Silesia (mozilla), Silesia (mr), Silesia (nci), Silesia (ooffice), Silesia (osdb), Silesia (reymont), Silesia (samba), Silesia (sao), Silesia (webster), Silesia (xml), and Silesia (x-ray) versus the bit rate values of the $\delta$-Huffman (Reconstruction with an Exception Code).

It can be noted that the $\delta$-Huffman (Dynamic Probability P2/Slice) use of the formula P2 and a NYT that starts with and maintains a value of 0 while the $\delta$-Huffman (Dynamic Probability P1/Slice) use of the formula P1 and a NYT that starts with a value of 1 and decreases over time, leads the $\delta$-Huffman (Dynamic Probability P2/Slice) to generate better bit rate values relative to entropy versus the $\delta$-Huffman (Dynamic Probability P1/Slice) for the Silesia data set.

The final note for experiment 9c is that the $\delta$-Huffman (Dynamic Probability P2/Slice) additional use of the slice method leads it to generate a better bit rate value relative to entropy for the Silesia (xml) file and similar bit rate values relative to entropy for the other files in the Silesia data set versus the $\delta$-Huffman (Dynamic Probability P2).

## 5.10 Experiment 10: δ-Huffman (Dynamic Probability P1/Slice/Reset)

Experiment 10 is divided into three parts: part 10a explains the compression of the synthetic data (integers), part 10b explains the compression of real-world data from the sorted inverted index gaps from Wikipedia, and part 10c explains the compression of the benchmark data set Silesia.

**5.10.1 Experiment 10a: Synthetic Data Set (integers).** $\delta$-Huffman (Dynamic Probability P1/Slice/Reset) estimates the probability function based on the use of a special symbol that represents the NYT and starts with a value of 1 and decreases over time, use of the formula P1, segmentation of the input data into slices $(L)$, and after the Huffman tree is updated at the end of a slice L, resetting the $\sum_0^i n$ value in the formula P1 to zero and the NYT value back to one. Figure 5.21 compares the $\delta$-Huffman (Dynamic Probability P1/Slice/Reset) bit rate values to the entropy values of the data set.

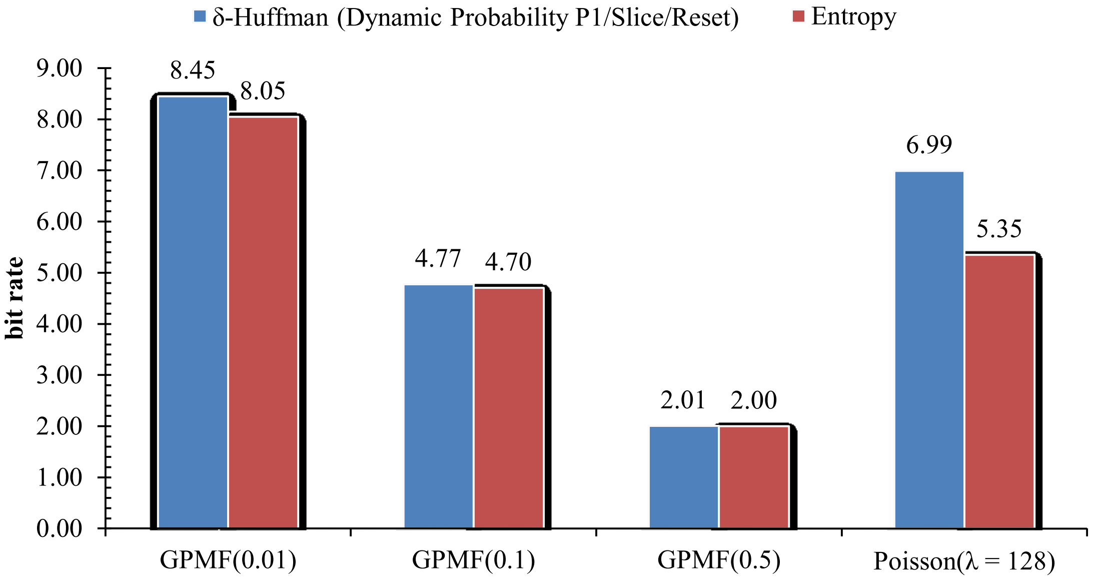


**Figure 5.21: δ-Huffman (Dynamic Probability P1/Slice/Reset) bit rate values vs. Entropy**

From figure 5.21 it can be noted that, for $\delta$-Huffman (Dynamic Probability P1/Slice/Reset), the average bit rate values differ from their respective estimated entropy values from a low difference of 0.01-bit for GPMF (0.5) to a high difference of 1.64 bits for Poisson (λ = 128).

From this experiment it can be noted that the $\delta$-Huffman (Dynamic Probability P1/Slice/Reset) generates bit rate values that are higher than the entropy; 0.40-bit more for GPMF (0.01), 0.07-bit more for GPMF (0.1), 0.01-bit more for GPMF (0.5), and 1.64 bits more for Poisson (λ=128) relative to the entropy of the synthetic data set of integers.

It can be noted that the $\delta$-Huffman (Dynamic Probability P1/Slice/Reset) assumptions for the estimation of the probability function cause it to:

- Generate lower bit rate values; 0.40-bit fewer for GPMF (0.01) and 0.04-bit fewer for GPMF (0.1), identical bit rate value for GPMF (0.5), and a higher bit rate value of 1.55 bits more for Poisson (λ = 128) versus the bit rate values of the $\delta$-Huffman (Dynamic Probability P2/Slice).

- Generate identical bit rate values for GPMF (0.01), GPMF (0.1), GPMF (0.5), and Poisson (λ = 128) versus the bit rate values of the $\delta$-Huffman (Dynamic Probability P1/Slice).
- Generate lower bit rate values; 0.38-bit fewer for GPMF (0.01) and 0.04-bit fewer for GPMF (0.1), identical bit rate value for GPMF (0.5), and a higher bit rate value of 1.55 bits more for Poisson (λ = 128) versus the bit rate values of the $\delta$-Huffman (Dynamic Probability P2).
- Generate identical bit rate values for GPMF (0.01), GPMF (0.1), and GPMF (0.5) and a higher bit rate value of 1.40 for Poisson (λ = 128) versus the bit rate values of the $\delta$-Huffman (Sibling/Static Probability).
- Generate lower bit rate values; 2.02 bits fewer for GPMF (0.1), 2.06 bits fewer for GPMF (0.5), and 0.01-bit fewer for Poisson (λ = 128) and an identical bit rate value for GPMF (0.01) versus the bit rate values of the $\delta$-Huffman (Static Probability).
- Generate lower bit rate values; 0.43-bit fewer for GPMF (0.01) and 0.04-bit fewer for GPMF (0.1), identical bit rate value for GPMF (0.5), and a higher bit rate value of 1.54 bits more for Poisson (λ = 128) versus the bit rate values of the $\delta$-Huffman (Update using Sibling Property with an Exception Code).
- Generate lower bit rate values; 0.38-bit fewer for GPMF (0.01) and 0.04-bit fewer for GPMF (0.1), identical bit rate value for GPMF (0.5), and a higher bit rate value of 1.55 bits more for Poisson (λ = 128) versus the bit rate values of the $\delta$-Huffman (Reconstruction with an Exception Code).

- Generate lower bit rate values; 0.81-bit fewer for GPMF (0.01), 0.97-bit fewer for GPMF (0.1), and 1.00-bit fewer for GPMF (0.5) and a higher bit rate value of 0.59-bit more for Poisson ($\lambda = 128$) versus the bit rate values of the $\delta$-Huffman (n+1).

It can be noted that the $\delta$-Huffman (Dynamic Probability P1/Slice/Reset) generates similar bit rate values relative to entropy versus the $\delta$-Huffman (Dynamic Probability P1/Slice). The $\delta$-Huffman (Dynamic Probability P1/Slice/Reset) additional use of the reset method does not generate any significant changes to the bit rate values relative to entropy for the synthetic data.

Note that the $\delta$-Huffman (Dynamic Probability P1/Slice/Reset), similar to the $\delta$-Huffman (Dynamic Probability P1/Slice) in the previous experiment 9a, generates better bit rate values relative to entropy for GPMF (0.5), GPMF (0.1), and GPMF (0.01) while the $\delta$-Huffman (Dynamic Probability P2/Slice) generates better bit rate values relative to entropy for Poisson.

The final note for experiment 10a is that the $\delta$-Huffman (Dynamic Probability P1/Slice/Reset) has the same strengths and weaknesses as the $\delta$-Huffman (Dynamic Probability P1/Slice) had with the other $\delta$-Huffman algorithms with experiments with the synthetic data set.

**5.10.2 Experiment 10b: Real-world Data (Wikipedia).** Figure 5.22 compares the $\delta$-Huffman (Dynamic Probability P1/Slice/Reset) bit rate values to the entropy values of the data set.

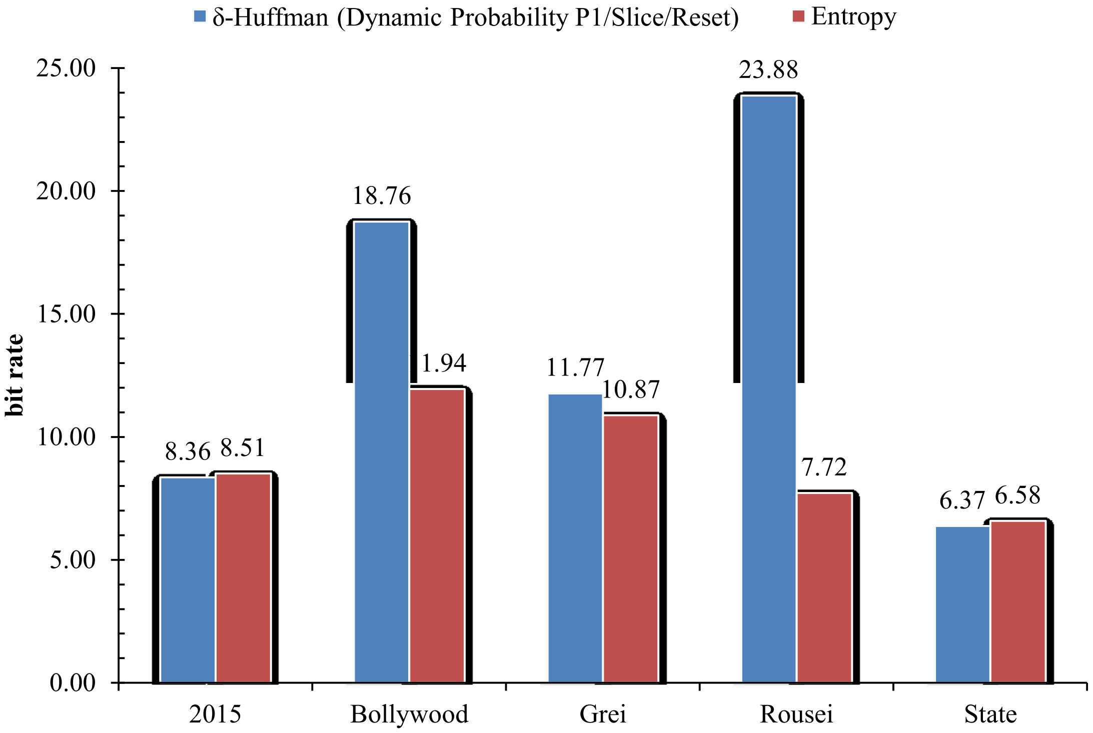


**Figure 5.22: δ-Huffman (Dynamic Probability P1/Slice/Reset) bit rate values vs. Entropy**

From figure 5.22 it can be noted that, for $\delta$-Huffman (Dynamic Probability P1/Slice/Reset), the bit rate values differ from their respective entropy values from a better than entropy difference of 0.21-bit for Wikipedia (State) to a high difference of 16.16 bits for Wikipedia (Rousei). These better bit rate values relative to entropy are further evaluated in Chapter 6. For now, only comparisons of the bit rate values relative to entropy results for the $\delta$-Huffman variants are discussed.

From this experiment it can be noted that the $\delta$-Huffman (Dynamic Probability P1/Slice/Reset) generates a better bit rate value relative to the entropy of the real-world data from the sorted inverted index gaps from Wikipedia for the two largest files, 0.15-bit fewer for Wikipedia (2015) and 0.21-bit fewer for Wikipedia (State) than for the smaller files with the larger difference of 0.89-bit more for Wikipedia (Grei), 6.81 bits more for Wikipedia (Bollywood), and 16.16 bits more for Wikipedia (Rousei).

It can be noted that the $\delta$-Huffman (Dynamic Probability P1/Slice/Reset) assumptions for the estimation of the probability function cause it to:

- Generate lower bit rate values; 0.29-bit fewer for Wikipedia (2015), 3.40 bits fewer for Wikipedia (Bollywood), 1.16 bits fewer for Wikipedia (Grei), 5.64 bits fewer for Wikipedia (Rousei), and 0.25-bit fewer for Wikipedia (State) versus the bit rate values of the $\delta$-Huffman (Dynamic Probability P2/Slice).
- Generate lower bit rate values; 0.38-bit fewer for Wikipedia (2015), 0.22-bit fewer for Wikipedia (Grei), and 0.39-bit fewer for Wikipedia (State), identical bit rate value for Wikipedia (Rousei), and a higher bit rate value of 2.44 bits more for Wikipedia (Bollywood) versus the bit rate values of the $\delta$-Huffman (Dynamic Probability P1/Slice).
- Generate lower bit rate values; 0.29-bit fewer for Wikipedia (2015), 3.41 bits fewer for Wikipedia (Bollywood), 1.14 bits fewer for Wikipedia (Grei), 7.43 bits fewer for Wikipedia (Rousei), and 0.25-bit fewer for Wikipedia (State) versus the bit rate values of the $\delta$-Huffman (Dynamic Probability P2).
- Generate lower bit rate values; 0.53-bit fewer for Wikipedia (2015), 6.01 bits fewer for Wikipedia (Bollywood), 6.63 bits fewer for Wikipedia (Grei), 1.63 bits fewer for Wikipedia (Rousei), and 0.90-bit fewer for Wikipedia (State) versus the bit rate values of the $\delta$-Huffman (Sibling/Static Probability).
- Generate lower bit rate values; 0.53-bit fewer for Wikipedia (2015), 6.00 bits fewer for Wikipedia (Bollywood), 6.63 bits fewer for Wikipedia (Grei), 1.63 bits fewer for Wikipedia (Rousei), and 0.90-bit fewer for Wikipedia (State) versus the bit rate values of the $\delta$-Huffman (Static Probability).

- Generate lower bit rate values; 0.30-bit fewer for Wikipedia (2015), 3.96 bits fewer for Wikipedia (Bollywood), 1.25 bits fewer for Wikipedia (Grei), 105.18 bits fewer for Wikipedia (Rousei), and 0.25-bit fewer for Wikipedia (State) versus the bit rate values of the $\delta$-Huffman (Update using the Sibling Property with an Exception Code).

It can be noted that the $\delta$-Huffman (Dynamic Probability P1/Slice/Reset) with the addition of the reset method generates better bit rate values relative to entropy for Wikipedia (2015), Wikipedia (State), and Wikipedia (Grei), a similar bit rate value relative to entropy for Wikipedia (Rousei), and a worse bit rate value relative to entropy for Wikipedia (Bollywood) versus the $\delta$-Huffman (Dynamic Probability P1/Slice).

The final note for experiment 10b is that the $\delta$-Huffman (Dynamic Probability P1/Slice/Reset) generates better bit rate values relative to entropy for all Wikipedia files versus the $\delta$-Huffman (Dynamic Probability P2/Slice), the $\delta$-Huffman (Dynamic Probability P2), the $\delta$-Huffman (Sibling/Static Probability), the $\delta$-Huffman (Static Probability), and the $\delta$-Huffman (Update using the Sibling Property with an Exception Code) for the Wikipedia data set.

**5.10.3 Experiment 10c: Benchmark Data Set (Silesia).** Figure 5.23 compares the $\delta$-Huffman (Dynamic Probability P1/Slice/Reset) bit rate values to the entropy values of the data set.

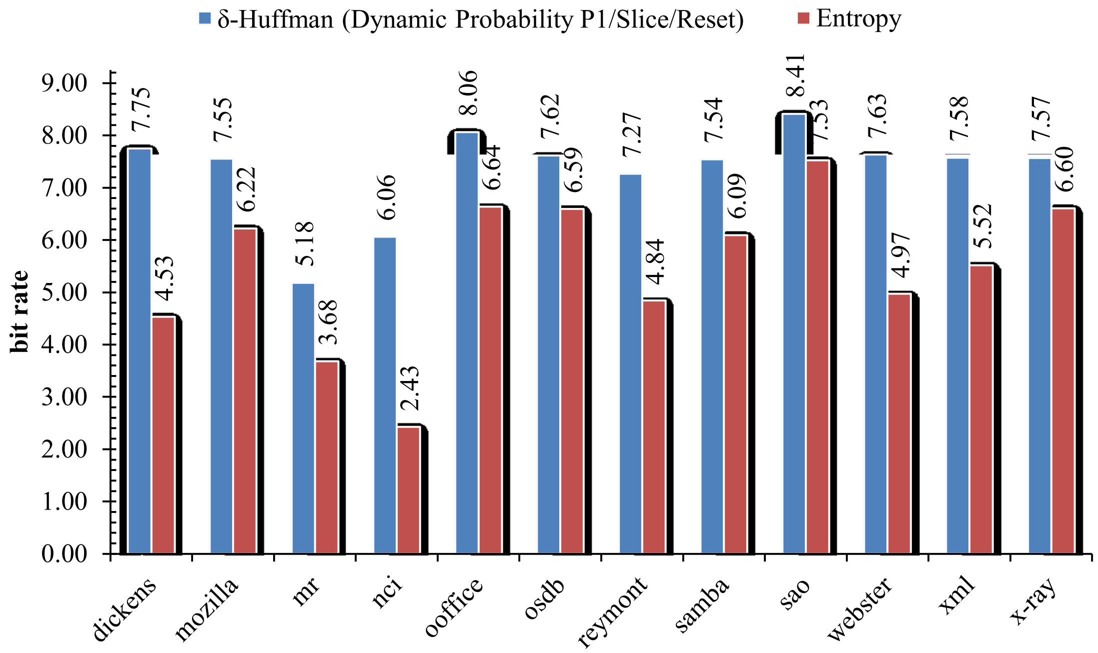


**Figure 5.23: δ-Huffman (Dynamic Probability P1/Slice/Reset) bit rate values vs. Entropy**

From figure 5.23 it can be noted that, for $\delta$-Huffman (Dynamic Probability P1/Slice/Reset), the bit rate values differ from their respective entropy values from a low difference of 0.89-bit for Silesia (sao) to a high difference of 3.63 bits for Silesia (nci).

From this experiment it can be noted that the $\delta$-Huffman (Dynamic Probability P1/Slice/Reset) generates bit rate values that are higher than the entropy; 3.22 bits more for Silesia (dickens), 1.33 bits more for Silesia (mozilla), 1.50 bits more for Silesia (mr), 3.63 bits more for Silesia (nci), 1.42 bits more for Silesia (ooffice), 1.02 bits more for Silesia (osdb), 2.42 bits more for Silesia (reymont), 1.45 bits more for Silesia (samba), 0.89-bit more for Silesia (sao), 2.66 bits more for Silesia (webster), 2.06 bits more for Silesia (xml), and 0.96-bit more for Silesia (x-ray) relative to the entropy of the benchmark data set Silesia.

It can be noted that the $\delta$-Huffman (Dynamic Probability P1/Slice/Reset) assumptions for the estimation of the probability function cause it to:

- Generate higher bit rate values; 3.18 bits more for Silesia (dickens), 1.31 bits more for Silesia (mozilla), 1.47 bits more for Silesia (mr), 3.63 bits more for Silesia (nci), 1.40 bits more for Silesia (ooffice), 1.00-bit more for Silesia (osdb), 2.40 bits more for Silesia (reymont), 1.42 bits more for Silesia (samba), 0.86-bit more for Silesia (sao), 2.63 bits more for Silesia (webster), 2.02 bits more for Silesia (xml), and 0.94-bit more for Silesia (x-ray) versus the bit rate values of the $\delta$-Huffman (Dynamic Probability P2/Slice).
- Generate lower bit rate values; 0.19-bit fewer for Silesia (mozilla), 0.85-bit fewer for Silesia (mr), 0.06-bit fewer for Silesia (ooffice), 0.07-bit fewer for Silesia (samba), 0.01-bit fewer for Silesia (sao), and 0.02-bit fewer for Silesia (xml), identical bit rate values for; Silesia (dickens), Silesia (nci), Silesia (osdb), and Silesia (reymont), and higher bit rate values; 0.01-bit more for Silesia (webster) and 0.14-bit more for Silesia (x-ray) versus the bit rate values of the $\delta$-Huffman (Dynamic Probability P1/Slice).
- Generate higher bit rate values; 3.18 bits more for Silesia (dickens), 1.31 bits more for Silesia (mozilla), 1.47 bits more for Silesia (mr), 3.63 bits more for Silesia (nci), 1.40 bits more for Silesia (ooffice), 1.00-bit more for Silesia (osdb), 2.40 bits more for Silesia (reymont), 1.42 bits more for Silesia (samba), 0.86-bit more for Silesia (sao), 2.63 bits more for Silesia (webster), 2.02 bits more for Silesia (xml), and 0.94-bit more for Silesia (x-ray) versus the bit rate values of the $\delta$-Huffman (Dynamic Probability P2).
- Generate higher bit rate values; 3.18 bits more for Silesia (dickens), 1.31 bits more for Silesia (mozilla), 1.47 bits more for Silesia (mr), 3.63 bits more for

Silesia (nci), 1.40 bits more for Silesia (ooffice), 1.00-bit more for Silesia (osdb), 2.40 bits more for Silesia (reymont), 1.42 bits more for Silesia (samba), 0.86-bit more for Silesia (sao), 2.63 bits more for Silesia (webster), 2.02 bits more for Silesia (xml), and 0.94-bit more for Silesia (x-ray) versus the bit rate values of the $\delta$-Huffman (Reconstruction with an Exception Code).

It can be noted that the $\delta$-Huffman (Dynamic Probability P1/Slice/Reset) additional use of the reset method leads it to generate equal or better bit rate values relative to entropy for all Silesia files except the Silesia (webster) and Silesia (x-ray) files versus the $\delta$-Huffman (Dynamic Probability P1/Slice).

The final note for experiment 10c is that the $\delta$-Huffman (Dynamic Probability P1/Slice/Reset) has the same weakness as the $\delta$-Huffman (Dynamic Probability P1/Slice) versus the $\delta$-Huffman (Dynamic Probability P2/Slice), the $\delta$-Huffman (Dynamic Probability P2), and the $\delta$-Huffman (Reconstruction with an Exception Code) for the Silesia data set.

## 5.11 Experiment 11: δ-Huffman (Dynamic Probability P2/Slice/Reset)

Experiment 11 is divided into three parts: part 11a explains the compression of the synthetic data (integers), part 11b explains the compression of real-world data from the sorted inverted index gaps from Wikipedia, and part 11c explains the compression of the benchmark data set Silesia.

**5.11.1 Experiment 11a: Synthetic Data Set (integers).** $\delta$-Huffman (Dynamic Probability P2/Slice/Reset) estimates the probability function based on the use of a special symbol that represents the NYT and maintains a value of 0, use of the formula P2, segmentation of the input data into slices $(L)$, and after the Huffman tree is updated at the end of a slice L, resetting the values for all data currently in the AT list to 0. Figure 5.24 compares the $\delta$-Huffman (Dynamic Probability P2/Slice/Reset) bit rate values to the entropy values of the data set.

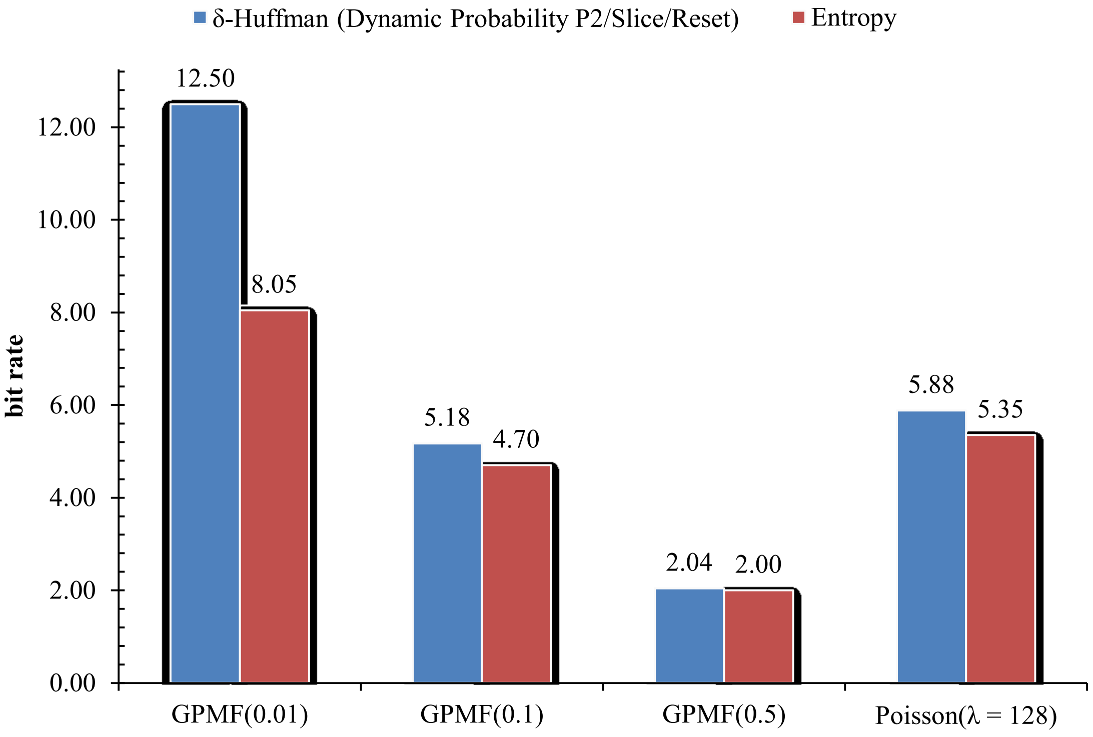


**Figure 5.24: δ-Huffman (Dynamic Probability P2/Slice/Reset) bit rate values vs. Entropy**

From figure 5.24 it can be noted that, for $\delta$-Huffman (Dynamic Probability P2/Slice/Reset), the bit rate values differ from their respective entropy values from a low difference of 0.04-bit for GPMF (0.5) to a high difference of 4.45 bits for GPMF (0.01).

From this experiment it can be noted that the $\delta$-Huffman (Dynamic Probability P2/Slice/Reset) generates bit rate values that are higher than the entropy; 4.45 bits more for GPMF (0.01), 0.47-bit more for GPMF (0.1), 0.04-bit more for GPMF (0.5), and 0.53-bit more for Poisson (λ=128) relative to the entropy of the synthetic data set of integers.

It can be noted that the $\delta$-Huffman (Dynamic Probability P2/Slice/Reset) assumptions for the estimation of the probability function cause it to:

- Generate a lower bit rate value of 1.11 bits fewer for Poisson (λ = 128) and higher bit rate values; 4.05 bits more for GPMF (0.01), 0.40-bit more for GPMF (0.1), and 0.04-bit more for GPMF (0.5) versus the bit rate values of the $\delta$-Huffman (Dynamic Probability P1/Slice/Reset).
- Generate higher bit rate values; 3.64 bits more for GPMF (0.01), 0.36-bit more for GPMF (0.1), 0.04-bit more for GPMF (0.5), and 0.45-bit more for Poisson (λ = 128) versus the bit rate values of the $\delta$-Huffman (Dynamic Probability P2/Slice).
- Generate lower bit rate values; 4.00 bits fewer for GPMF (0.01), 4.30 bits fewer for GPMF (0.1), 1.96 bits fewer for GPMF (0.5), and 6.46 bits fewer for Poisson (λ = 128) versus the bit rate values of the $\delta$-Huffman (Dynamic Probability P1/Slice).
- Generate higher bit rate values; 3.67 bits more for GPMF (0.01), 0.36-bit more for GPMF (0.1), 0.04-bit more for GPMF (0.5), and 0.44-bit more for Poisson (λ = 128) versus the bit rate values of the $\delta$-Huffman (Dynamic Probability P2).
- Generate higher bit rate values; 4.04 bits more for GPMF (0.01), 0.40-bit more for GPMF (0.1), 0.04-bit more for GPMF (0.5), and 0.29-bit more for Poisson (λ = 128) versus the bit rate values of the $\delta$-Huffman (Sibling/Static Probability).

- Generate lower bit rate values; 1.62 bits fewer for GPMF (0.1), 2.03 bits fewer for GPMF (0.5), and 1.12 bits fewer for Poisson (λ = 128) and a higher bit rate value of 4.04 bits more for GPMF (0.01) versus the bit rate values of the $\delta$-Huffman (Static Probability).
- Generate higher bit rate values; 3.62 bits more for GPMF (0.01), 0.36-bit more for GPMF (0.1), 0.03-bit more for GPMF (0.5), and 0.43-bit more for Poisson (λ = 128) versus the bit rate values of the $\delta$-Huffman (Update using Sibling Property with an Exception Code).
- Generate higher bit rate values; 3.67 bits more for GPMF (0.01), 0.36-bit more for GPMF (0.1), 0.03-bit more for GPMF (0.5), and 0.44-bit more for Poisson (λ = 128) versus the bit rate values of the $\delta$-Huffman (Reconstruction with an Exception Code).
- Generate lower bit rate values; 0.57-bit fewer for GPMF (0.1), 0.96-bit fewer for GPMF (0.5), and 0.52-bit fewer for Poisson (λ = 128) and a higher bit rate value of 3.24 bits more for GPMF (0.01) versus the bit rate values of the $\delta$-Huffman (n+1).

It can be noted that the $\delta$-Huffman (Dynamic Probability P2/Slice/Reset) additional use of the reset method leads it to generate worse bit rate values relative to entropy versus the $\delta$-Huffman (Dynamic Probability P2/Slice). It can be noted that the $\delta$-Huffman (Dynamic Probability P2/Slice/Reset) generates worse bit rate values relative to entropy versus the $\delta$-Huffman (Dynamic Probability P2).

Note that the $\delta$-Huffman (Dynamic Probability P2/Slice/Reset) use of the reset method cause it to generate a worse bit rate value relative to entropy for the GPMF (0.01) file versus the $\delta$-Huffman (n+1).

The final note for experiment 11a is that, overall, the additional use of the reset method by the $\delta$-Huffman (Dynamic Probability P2/Slice/Reset) does not generate better bit rate values relative to entropy results for the synthetic data versus the other $\delta$-Huffman algorithms used in the experiments for the synthetic data set.

**5.11.2 Experiment 11b: Real-world Data (Wikipedia).** Figure 5.25 compares the $\delta$-Huffman (Dynamic Probability P2/Slice/Reset) bit rate values to the entropy values of the data set.

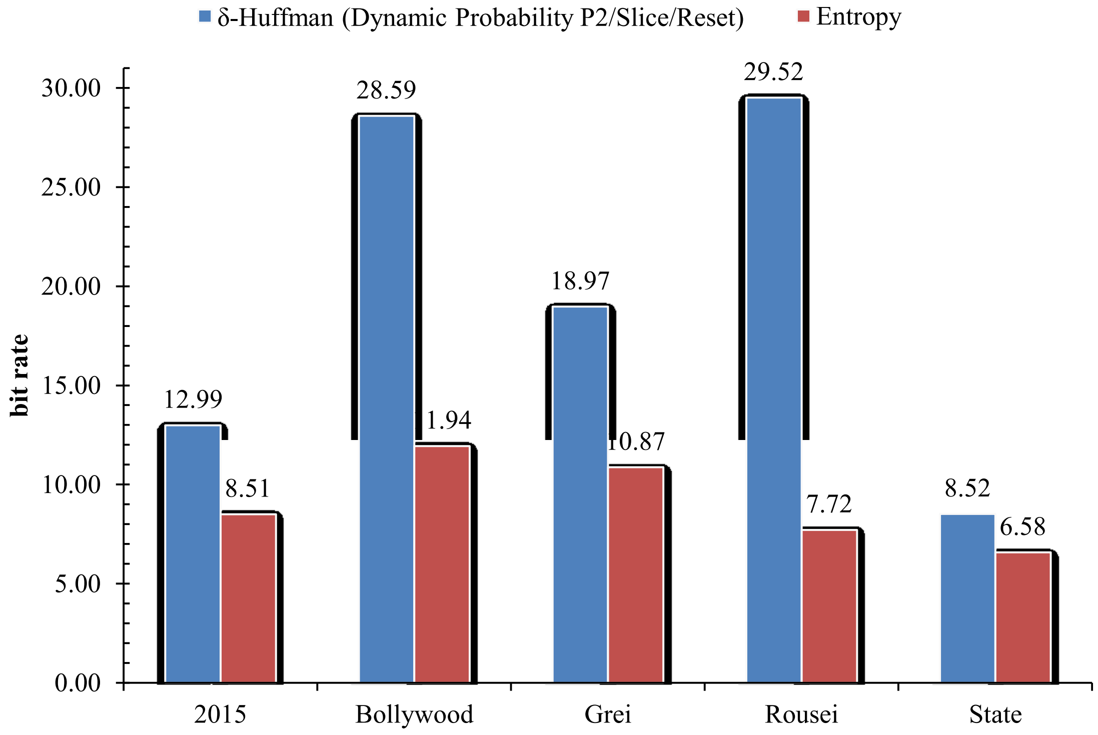


**Figure 5.25: δ-Huffman (Dynamic Probability P2/Slice/Reset) bit rate values vs. Entropy**

From figure 5.25 it can be noted that, for $\delta$-Huffman (Dynamic Probability P2/Slice/Reset), the bit rate values differ from their respective entropy values from a low difference of 1.94 bits for Wikipedia (State) to a high difference of 21.80 bits for Wikipedia (Rousei).

From this experiment it can be noted that the $\delta$-Huffman (Dynamic Probability P2/Slice/Reset) generates a higher bit rate value relative to the entropy of the real-world data from the sorted inverted index gaps from Wikipedia. The δ-Huffman (Dynamic Probability P2/Slice/Reset) for larger files generates the relatively large difference of 4.49 bits more for Wikipedia (2015) and 1.94 bits more for Wikipedia (State) than for the smaller files with the relatively larger difference of 8.10 bits more for Wikipedia (Grei), 16.65 bits more for Wikipedia (Bollywood), and 21.80 bits more for Wikipedia (Rousei).

It can be noted that the $\delta$-Huffman (Dynamic Probability P2/Slice/Reset) assumptions for the estimation of the probability function cause it to:

- Generate higher bit rate values; 4.64 bits more for Wikipedia (2015), 9.83 bits more for Wikipedia (Bollywood), 7.20 bits more for Wikipedia (Grei), 5.64 bits more for Wikipedia (Rousei), and 2.15 bits more for Wikipedia (State) versus the bit rate values of the $\delta$-Huffman (Dynamic Probability P1/Slice/Reset).
- Generate an identical bit rate value for Wikipedia (Rousei) and higher bit rate values; 4.34 bits more for Wikipedia (2015), 6.44 bits more for Wikipedia (Bollywood), 6.04 bits more for Wikipedia (Grei), and 1.90 bits more for Wikipedia (State) versus the bit rate values of the $\delta$-Huffman (Dynamic Probability P2/Slice).
- Generate higher bit rate values; 4.26 bits more for Wikipedia (2015), 12.27 bits more for Wikipedia (Bollywood), 6.98 bits more for Wikipedia (Grei),

5.64 bits more for Wikipedia (Rousei), and 1.76 bits more for Wikipedia (State) versus the bit rate values of the $\delta$-Huffman (Dynamic Probability P1/Slice).

- Generate a lower bit rate value of 1.79 bits fewer for Wikipedia (Rousei) and higher bit rate values; 4.35 bits more for Wikipedia (2015), 6.43 bits more for Wikipedia (Bollywood), 6.06 bits more for Wikipedia (Grei), and 1.90 bits more for Wikipedia (State) versus the bit rate values of the $\delta$-Huffman (Dynamic Probability P2).
- Generate higher bit rate values; 4.11 bits more for Wikipedia (2015), 3.83 bits more for Wikipedia (Bollywood), 0.58-bit more for Wikipedia (Grei), 4.00 bits more for Wikipedia (Rousei), and 1.25 bits more for Wikipedia (State) versus the bit rate values of the $\delta$-Huffman (Sibling/Static Probability).
- Generate higher bit rate values; 4.11 bits more for Wikipedia (2015), 3.83 bits more for Wikipedia (Bollywood), 0.58-bit more for Wikipedia (Grei), 4.00 bits more for Wikipedia (Rousei), and 1.25 bits more for Wikipedia (State) versus the bit rate values of the $\delta$-Huffman (Static Probability).
- Generate a lower bit rate value of 99.55 bits fewer for Wikipedia (Rousei) and higher bit rate values; 4.34 bits more for Wikipedia (2015), 5.87 bits more for Wikipedia (Bollywood), 5.96 bits more for Wikipedia (Grei), and 1.90 bits more for Wikipedia (State) versus the bit rate values of the $\delta$-Huffman (Update using the Sibling Property with an Exception Code).

The final note for experiment 11b is that similar to the previous experiment 11a, the $\delta$-Huffman (Dynamic Probability P2/Slice/Reset) additional use of the reset method leads it to generate the worse bit rate values relative to entropy results for the Wikipedia data versus the other $\delta$-Huffman algorithms used in the experiments for the real-world data set, with the one exception of a better bit rate value relative to entropy for the Wikipedia (Rousei) file versus the $\delta$-Huffman (Update using the Sibling Property with an Exception Code).

**5.11.3 Experiment 11c: Benchmark Data Set (Silesia).** Figure 5.26 compares the $\delta$-Huffman (Dynamic Probability P2/Slice/Reset) bit rate values to the entropy values of the data set.

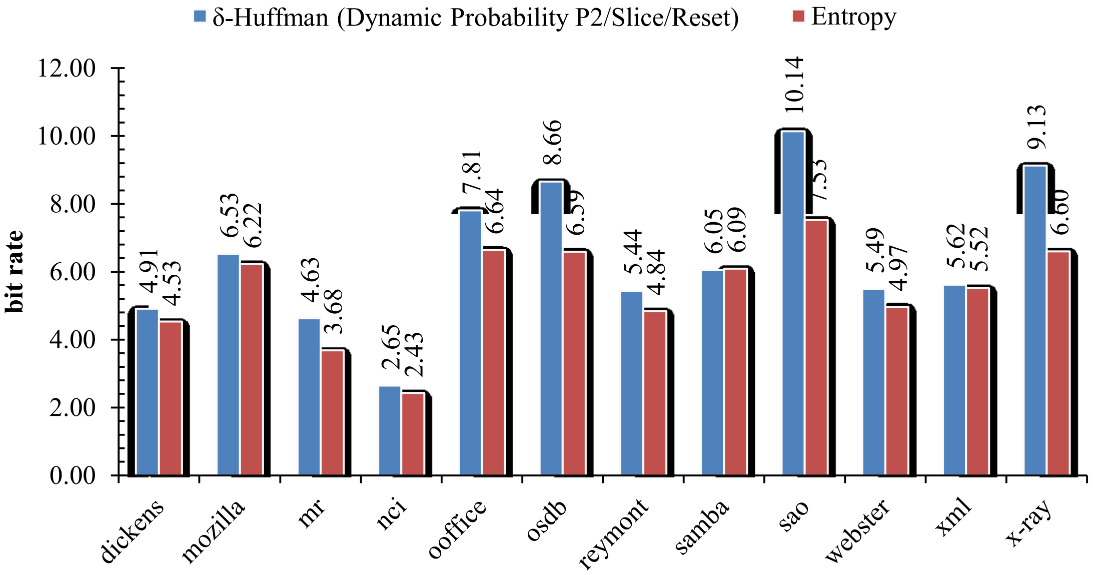


**Figure 5.26: δ-Huffman (Dynamic Probability P2/Slice/Reset) bit rate values vs. Entropy**

From figure 5.26 it can be noted that, for $\delta$-Huffman (Dynamic Probability P2/Slice/Reset), the bit rate values differ from their respective entropy values from a better than entropy difference of 0.04-bit for Silesia (samba) to a high difference of 2.61 bits for Silesia (sao).

From this experiment it can be noted that the $\delta$-Huffman (Dynamic Probability P2/Slice/Reset) generates a lower bit rate value of 0.04-bit fewer for Silesia (samba) and higher bit rate values; 0.38-bit more for Silesia (dickens), 0.31-bit more for Silesia (mozilla), 0.95-bit more for Silesia (mr), 0.22-bit more for Silesia (nci), 1.17 bits more for Silesia (ooffice), 2.06 bits more for Silesia (osdb), 0.59-bit more for Silesia (reymont), 2.61 bits more for Silesia (sao), 0.51-bit more for Silesia (webster), 0.10-bit more for Silesia (xml), and 2.53 bits more for Silesia (x-ray) relative to the entropy of the benchmark data set Silesia.

It can be noted that the $\delta$-Huffman (Dynamic Probability P2/Slice/Reset) assumptions for the estimation of the probability function cause it to:

- Generate lower bit rate values; 2.84 bits fewer for Silesia (dickens), 1.02 bits fewer for Silesia (mozilla), 0.55-bit fewer for Silesia (mr), 3.42 bits fewer for Silesia (nci), 0.25-bit fewer for Silesia (ooffice), 1.83 bits fewer for Silesia (reymont), 1.49 bits fewer for Silesia (samba), 2.15 bits fewer for Silesia (webster), and 1.96 bits fewer for Silesia (xml) and higher bit rate values; 1.04 bits more for Silesia (osdb), 1.72 bits more for Silesia (sao), and 1.56 bits more for Silesia (x-ray) versus the bit rate values of the $\delta$-Huffman (Dynamic Probability P1/Slice/Reset).
- Generate a lower bit rate value of 0.07-bit fewer for Silesia (samba) and higher bit rate values; 0.34-bit more for Silesia (dickens), 0.28-bit more for Silesia (mozilla), 0.92-bit more for Silesia (mr), 0.21-bit more for Silesia (nci), 1.15 bits more for Silesia (ooffice), 2.04 bits more for Silesia (osdb), 0.57-bit more for Silesia (reymont), 2.59 bits more for Silesia (sao), 0.48-bit more for Silesia (webster), 0.06-bit more for Silesia (xml), and 2.50 bits more

for Silesia (x-ray) versus the bit rate values of the $\delta$-Huffman (Dynamic Probability P2/Slice).

- Generate lower bit rate values; 2.84 bits fewer for Silesia (dickens), 1.22 bits fewer for Silesia (mozilla), 1.40 bits fewer for Silesia (mr), 3.41 bits fewer for Silesia (nci), 0.31-bit fewer for Silesia (ooffice), 1.83 bits fewer for Silesia (reymont), 1.56 bits fewer for Silesia (samba), 2.14 bits fewer for Silesia (webster), and 1.98 bits fewer for Silesia (xml) and higher bit rate values; 1.04 bits more for Silesia (osdb), 1.71 bits more for Silesia (sao), and 1.70 bits more for Silesia (x-ray) versus the bit rate values of the $\delta$-Huffman (Dynamic Probability P1/Slice).
- Generate a lower bit rate value of 0.07-bit fewer for Silesia (samba) and higher bit rate values; 0.34-bit more for Silesia (dickens), 0.28-bit more for Silesia (mozilla), 0.92-bit more for Silesia (mr), 0.21-bit more for Silesia (nci), 1.15 bits more for Silesia (ooffice), 2.04 bits more for Silesia (osdb), 0.57-bit more for Silesia (reymont), 2.59 bits more for Silesia (sao), 0.48-bit more for Silesia (webster), 0.06-bit more for Silesia (xml), and 2.50 bits more for Silesia (x-ray) versus the bit rate values of the $\delta$-Huffman (Dynamic Probability P2).
- Generate a lower bit rate value of 0.07-bit fewer for Silesia (samba) and higher bit rate values; 0.34-bit more for Silesia (dickens), 0.28-bit more for Silesia (mozilla), 0.92-bit more for Silesia (mr), 0.21-bit more for Silesia (nci), 1.15 bits more for Silesia (ooffice), 2.04 bits more for Silesia (osdb), 0.57-bit more for Silesia (reymont), 2.59 bits more for Silesia (sao), 0.48-bit

more for Silesia (webster), 0.06-bit more for Silesia (xml), and 2.50 bits more for Silesia (x-ray) versus the bit rate values of the $\delta$-Huffman (Reconstruction with an Exception Code).

It can be noted that the $\delta$-Huffman (Dynamic Probability P2/Slice/Reset) generates better bit rate values relative to entropy, with the exception of the Silesia (sao) and Silesia (x-ray) files, versus the $\delta$-Huffman (Dynamic Probability P1/Slice/Reset).

The final note for experiment 11c is that, based on the bit rate values relative to entropy results for all $\delta$-Huffman algorithms from this and previous experiments with the Silesia data set, that overall the addition of the slice method or the slice and reset methods for certain $\delta$-Huffman algorithms does not make any noticeable significant improvement to the bit rate values relative to entropy versus the $\delta$-Huffman (Dynamic Probability P2) and the $\delta$-Huffman (Reconstruction with an Exception Code) for the Silesia data set.

## 5.12 Experiment 12: δ-Huffman (FLC)

Experiment 12 is divided into three parts: part 12a explains the compression of the synthetic data (bytes), part 12b explains the compression of real-world data from the sorted inverted index gaps from Wikipedia (bytes), and part 12c explains the compression of the benchmark data set Silesia.

**5.12.1 Experiment 12a: Synthetic Data Set (bytes).** $\delta$-Huffman (FLC) estimates the probability function based on the use of the formula $p_n$ at the end of an iteration. Figure 5.27 compares the $\delta$-Huffman (FLC) bit rate values to the entropy values of the data set.

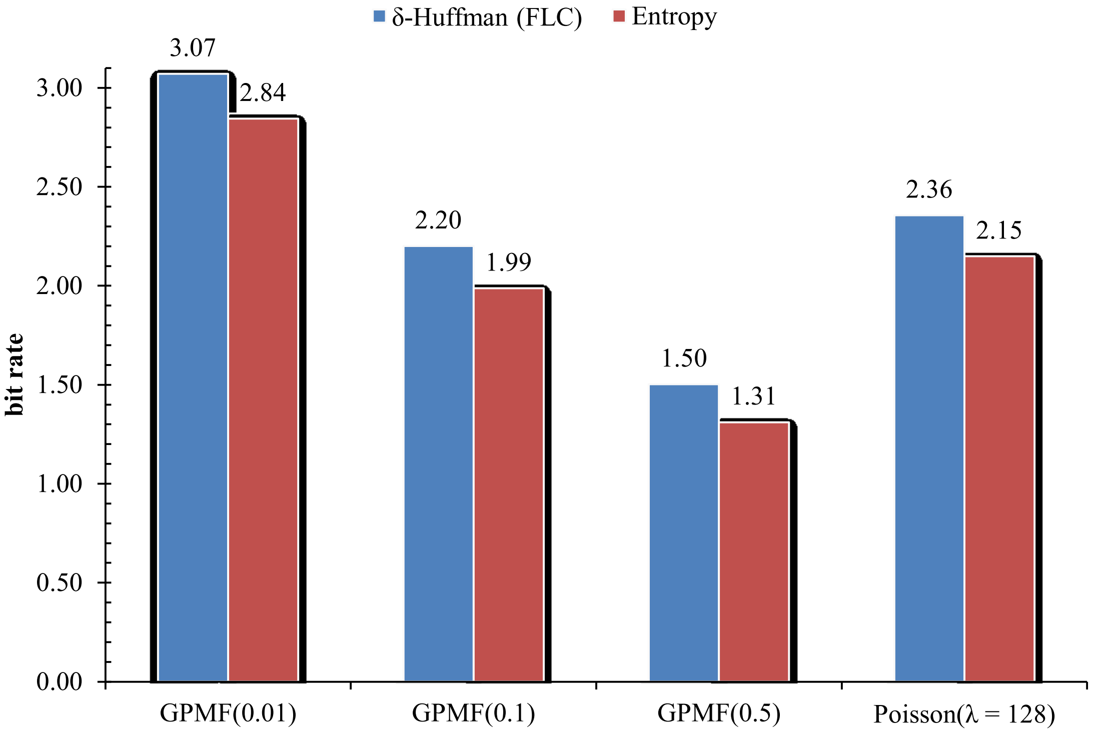


**Figure 5.27: δ-Huffman (FLC) bit rate values (bytes) vs. Entropy**

From figure 5.27 it can be noted that, for $\delta$-Huffman (FLC), the bit rate values differ from their respective entropy values from a low difference of 0.19-bit for GPMF (0.5) to a high difference of 0.23-bit for GPMF (0.01).

From this experiment it can be noted that the $\delta$-Huffman (FLC) generates bit rate values that are higher than the entropy; 0.23-bit more for GPMF (0.01), 0.21-bit more for GPMF (0.1), 0.19-bit more for GPMF (0.5), and 0.21-bit more for Poisson (λ=128) relative to the entropy of the synthetic data set of integers (bytes).

It can be noted that the $\delta$-Huffman (FLC) assumptions for the estimation of the probability function cause it to:

- Generate lower bit rate values; 0.03-bit fewer for GPMF (0.01) and 0.01-bit fewer for Poisson (λ=128) and identical bit rate values for GPMF (0.1) and

GPMF (0.5) versus the bit rate values of the $\delta$-Huffman (Reconstruction with an Exception Code).

- Generate lower bit rate values; 0.96-bit fewer for GPMF (0.01), 0.98-bit fewer for GPMF (0.1), 1.00-bit fewer for GPMF (0.5), and 0.99-bit fewer for Poisson (λ=128) versus the bit rate values of the $\delta$-Huffman (Flag/FLC).

It can be noted that the $\delta$-Huffman (FLC) variant generates better bit rate values relative to entropy for GPMF (0.1) and GPMF (0.01) versus the $\delta$-Huffman (Reconstruction with an Exception Code) variant.

The final note for experiment 12a is that the $\delta$-Huffman (FLC) variant generates better bit rate values relative to entropy versus the $\delta$-Huffman (Flag/FLC) variant, that uses a flag bit, for the synthetic data set of integers (bytes).

**5.12.2 Experiment 12b: Real-world Data (Wikipedia) (bytes).** Figure 5.28 compares the $\delta$-Huffman (FLC) bit rate values to the entropy values of the data set.

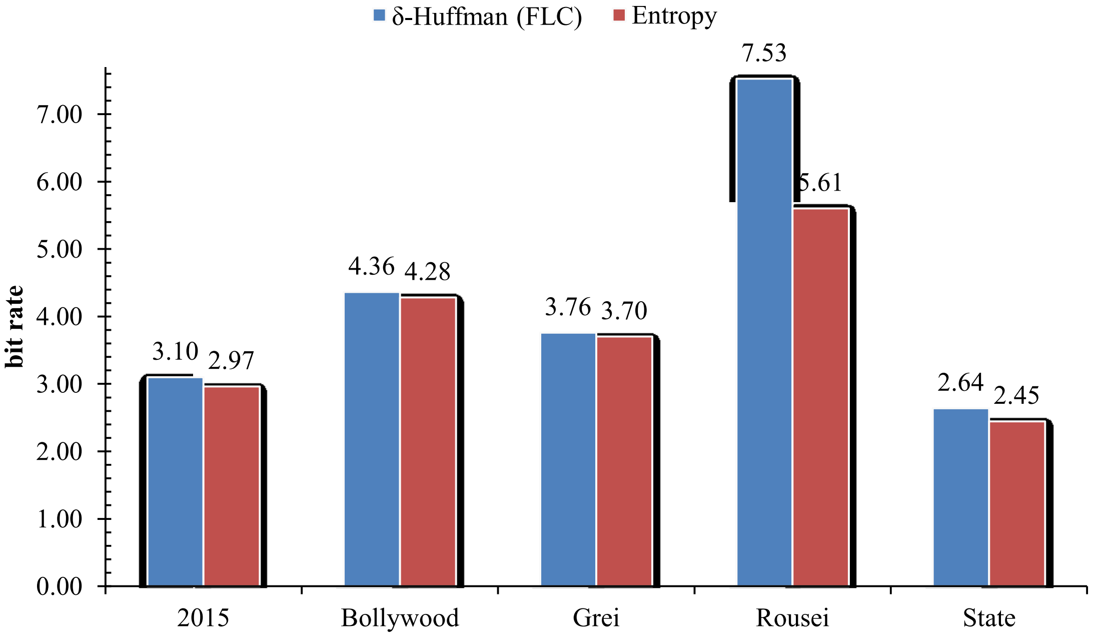


**Figure 5.28: δ-Huffman (FLC) bit rate values (bytes) vs. Entropy**

From figure 5.28 it can be noted that, for $\delta$-Huffman (FLC), the bit rate values differ from their respective entropy values from a low difference of 0.06-bit for Wikipedia (Grei) to a high difference of 1.92 bits for Wikipedia (Rousei).

The final note for experiment 12b is that the $\delta$-Huffman (FLC) generates a higher bit rate value relative to the entropy of the real-world data from the sorted inverted index gaps from Wikipedia; 0.19-bit more for Wikipedia (State), 0.14-bit more for Wikipedia (2015), 0.06-bit more for Wikipedia (Grei), 0.08-bit more for Wikipedia (Bollywood), and 1.92 bits more for Wikipedia (Rousei) relative to the entropy of the real-world data set Wikipedia handled as bytes.

**5.12.3 Experiment 12c: Benchmark Data Set (Silesia).** Figure 5.29 compares the $\delta$-Huffman (FLC) bit rate values to the entropy values of the data set.

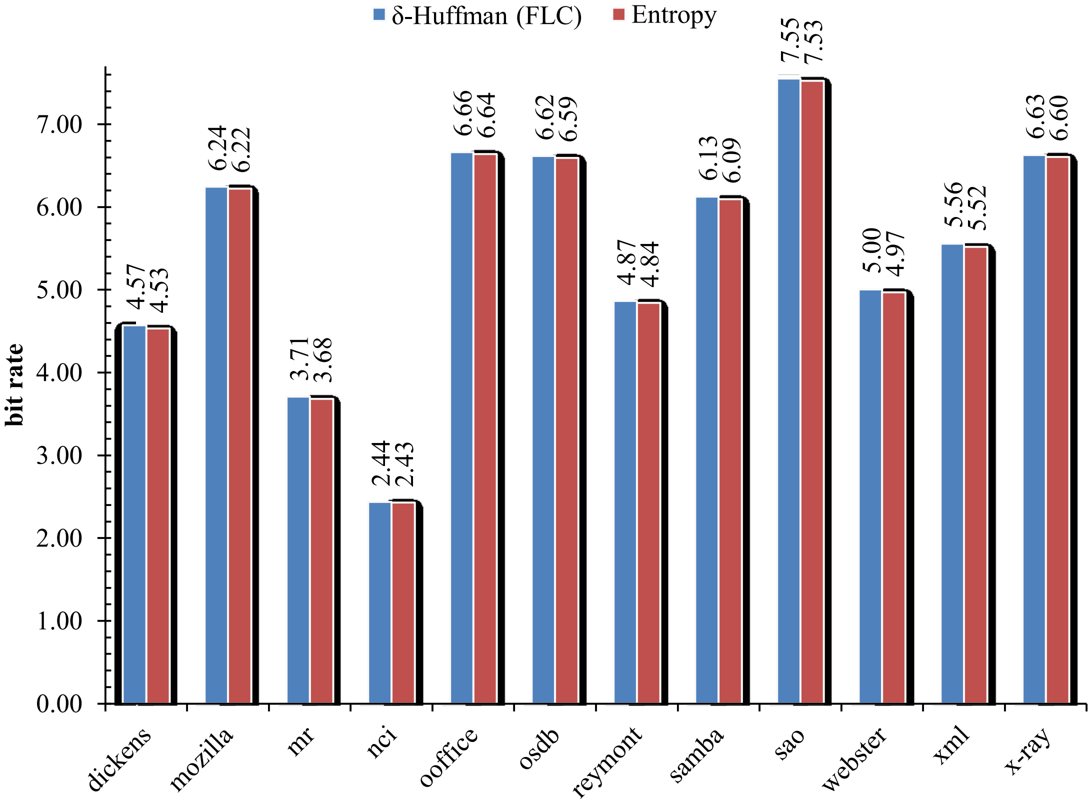


**Figure 5.29: δ-Huffman (FLC) bit rate values vs. Entropy**

From figure 5.29 it can be noted that, for $\delta$-Huffman (FLC), the bit rate values differ from their respective entropy values from a low difference of 0.01-bit for Silesia (nci) to a high difference of 0.04-bit for Silesia (dickens).

From this experiment it can be noted that the $\delta$-Huffman (FLC) generates bit rate values that are higher than the entropy; 0.04-bit more for Silesia (dickens), 0.02-bit more for Silesia (mozilla), 0.03-bit more for Silesia (mr), 0.01-bit more for Silesia (nci), 0.02-bit more for Silesia (ooffice), 0.02-bit more for Silesia (osdb), 0.02-bit more for Silesia (reymont), 0.03-bit more for Silesia (samba), 0.03-bit more for Silesia (sao), 0.03-bit more for Silesia (webster), 0.04-bit more for Silesia (xml), and 0.03-bit more for Silesia (x-ray) relative to the entropy of the benchmark data set Silesia.

It can be noted that the $\delta$-Huffman (FLC) assumptions for the estimation of the probability function cause it to:

- Generate lower bit rate values; 0.34-bit fewer for Silesia (dickens), 0.28-bit fewer for Silesia (mozilla), 0.92-bit fewer for Silesia (mr), 0.21-bit fewer for Silesia (nci), 1.15 bits fewer for Silesia (ooffice), 2.04 bits fewer for Silesia (osdb), 0.57-bit fewer for Silesia (reymont), 2.59 bits fewer for Silesia (sao), 0.48-bit fewer for Silesia (webster), 0.06-bit fewer for Silesia (xml), and 2.50 bits fewer for Silesia (x-ray) and a higher bit rate value of 0.07-bit more for Silesia (samba) versus the bit rate values of the $\delta$-Huffman (Dynamic Probability P2/Slice/Reset).
- Generate lower bit rate values; 3.18 bits fewer for Silesia (dickens), 1.31 bits fewer for Silesia (mozilla), 1.47 bits fewer for Silesia (mr), 3.63 bits fewer for Silesia (nci), 1.40 bits fewer for Silesia (ooffice), 1.00-bit fewer for Silesia

(osdb), 2.40 bits fewer for Silesia (reymont), 1.42 bits fewer for Silesia (samba), 0.86-bit fewer for Silesia (sao), 2.63 bits fewer for Silesia (webster), 2.02 bits fewer for Silesia (xml), and 0.94-bit fewer for Silesia (x-ray) versus the bit rate values of the $\delta$-Huffman (Dynamic Probability P1/Slice/Reset).

- Generate identical bit rate values for; Silesia (dickens), Silesia (mozilla), Silesia (mr), Silesia (nci), Silesia (ooffice), Silesia (osdb), Silesia (reymont), Silesia (samba), Silesia (sao), Silesia (webster), Silesia (xml), and Silesia (x-ray) versus the bit rate values of the $\delta$-Huffman (Dynamic Probability P2/Slice).
- Generate lower bit rate values; 3.18 bits fewer for Silesia (dickens), 1.50 bits fewer for Silesia (mozilla), 2.32 bits fewer for Silesia (mr), 3.62 bits fewer for Silesia (nci), 1.45 bits fewer for Silesia (ooffice), 1.00-bit fewer for Silesia (osdb), 2.40 bits fewer for Silesia (reymont), 1.49 bits fewer for Silesia (samba), 0.88-bit fewer for Silesia (sao), 2.62 bits fewer for Silesia (webster), 2.04 bits fewer for Silesia (xml), and 0.80-bit fewer for Silesia (x-ray) versus the bit rate values of the $\delta$-Huffman (Dynamic Probability P1/Slice).
- Generate identical bit rate values for; Silesia (dickens), Silesia (mozilla), Silesia (mr), Silesia (nci), Silesia (ooffice), Silesia (osdb), Silesia (reymont), Silesia (samba), Silesia (sao), Silesia (webster), Silesia (xml), and Silesia (x-ray) versus the bit rate values of the $\delta$-Huffman (Dynamic Probability P2).
- Generate identical bit rate values for; Silesia (dickens), Silesia (mozilla), Silesia (mr), Silesia (nci), Silesia (ooffice), Silesia (osdb), Silesia (reymont), Silesia (samba), Silesia (sao), Silesia (webster), Silesia (xml), and Silesia (x-

ray) versus the bit rate values of the $\delta$-Huffman (Reconstruction with an Exception Code).

The final note for experiment 12c is that the $\delta$-Huffman (FLC) generates bit rate values relative to entropy results that are similar to the better performing $\delta$-Huffman algorithms in experiment 11c, the $\delta$-Huffman (Reconstruction with an Exception Code) and the $\delta$-Huffman (Dynamic Probability P2) for the Silesia data set.

## 5.13 Conclusion

This section concludes Chapter 5 with figures that compare the $\delta$-Huffman variants bit rate value results relative to entropy for the synthetic data set, the real-world data set Wikipedia, and the benchmark data set Silesia.

**5.13.1 Results of data sets (integers).** Figures 5.30, 5.32, and 5.35 present the legend for their associated $\delta$-Huffman figures in this section. The bit rate values are on the horizontal axis and the file names are on the vertical axis. Figure 5.31 compares the bit rate values of the ten $\delta$-Huffman variants experiments with the synthetic data set handled as integers.

- 🟦 Entropy
- 🟥 δ-Huffman n+1
- 🟩 δ-Huffman Reconstruction with an Exception Code
- 🟪 δ-Huffman Update using Sibling Property with an Exception Code
- 🟦 δ-Huffman Static Probability
- 🟧 δ-Huffman Sibling/Static Probability
- 🟦 δ-Huffman Dynamic Probability P2
- 🟥 δ-Huffman Dynamic Probability P1/Slice
- 🟩 δ-Huffman Dynamic Probability P2/Slice
- 🟪 δ-Huffman Dynamic Probability P1/Slice/Reset
- 🟦 δ-Huffman Dynamic Probability P2/Slice/Reset

**Figure 5.30: Legend for Figure 5.31**

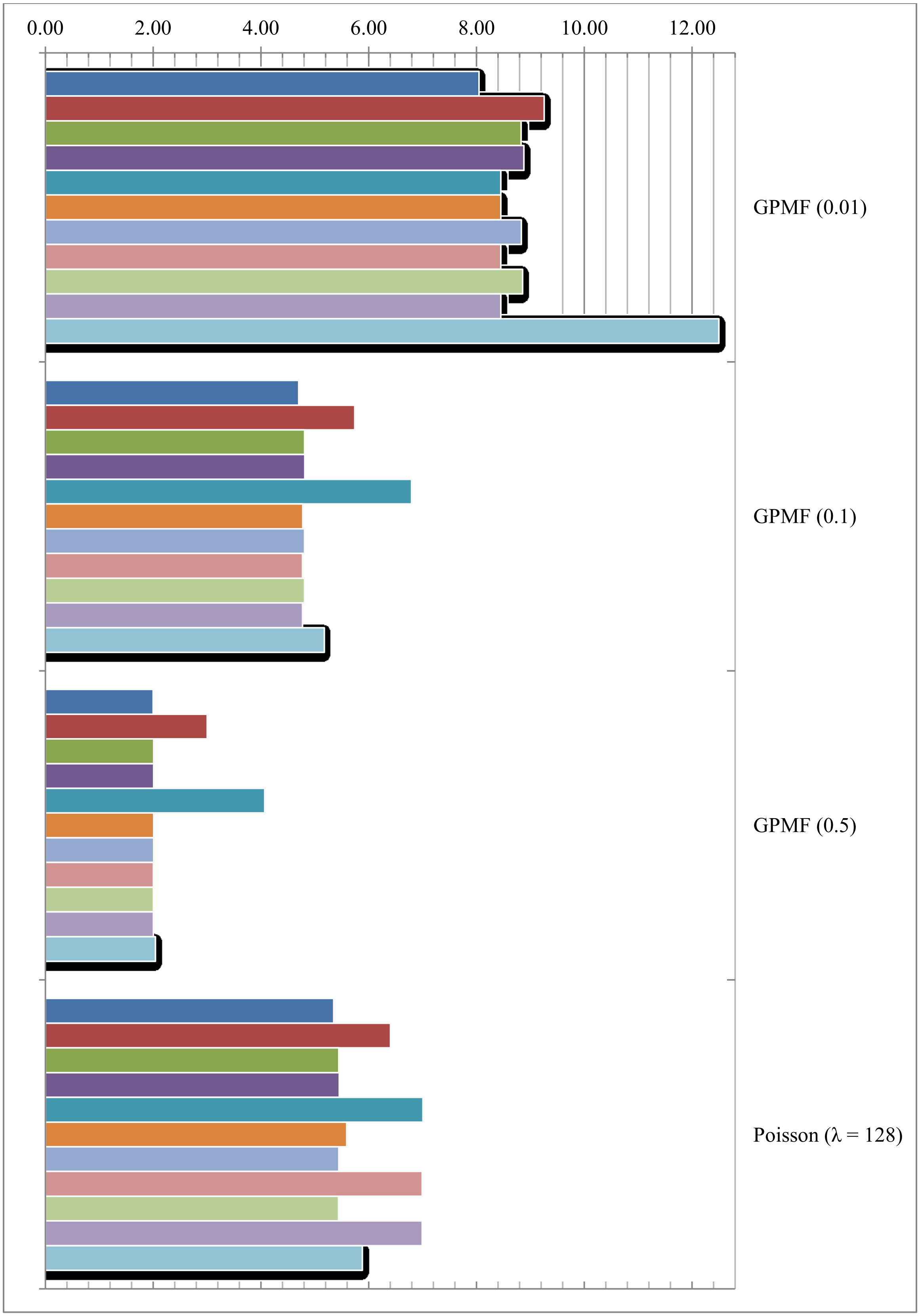


**Figure 5.31: δ-Huffman variants bit rate values vs. Entropy for the synthetic data set**

For ease of readability, figure 5.33 compares the bit rate values of the eight $\delta$-Huffman variants experiments with the real-world data set Wikipedia to the bit rate values of entropy with the Wikipedia (Rousei) results removed while figure 5.34 compares the bit rate values of the eight $\delta$-Huffman variants experiments with the real-world data set Wikipedia to the bit rate values of entropy with the Wikipedia (Rousei) results included.

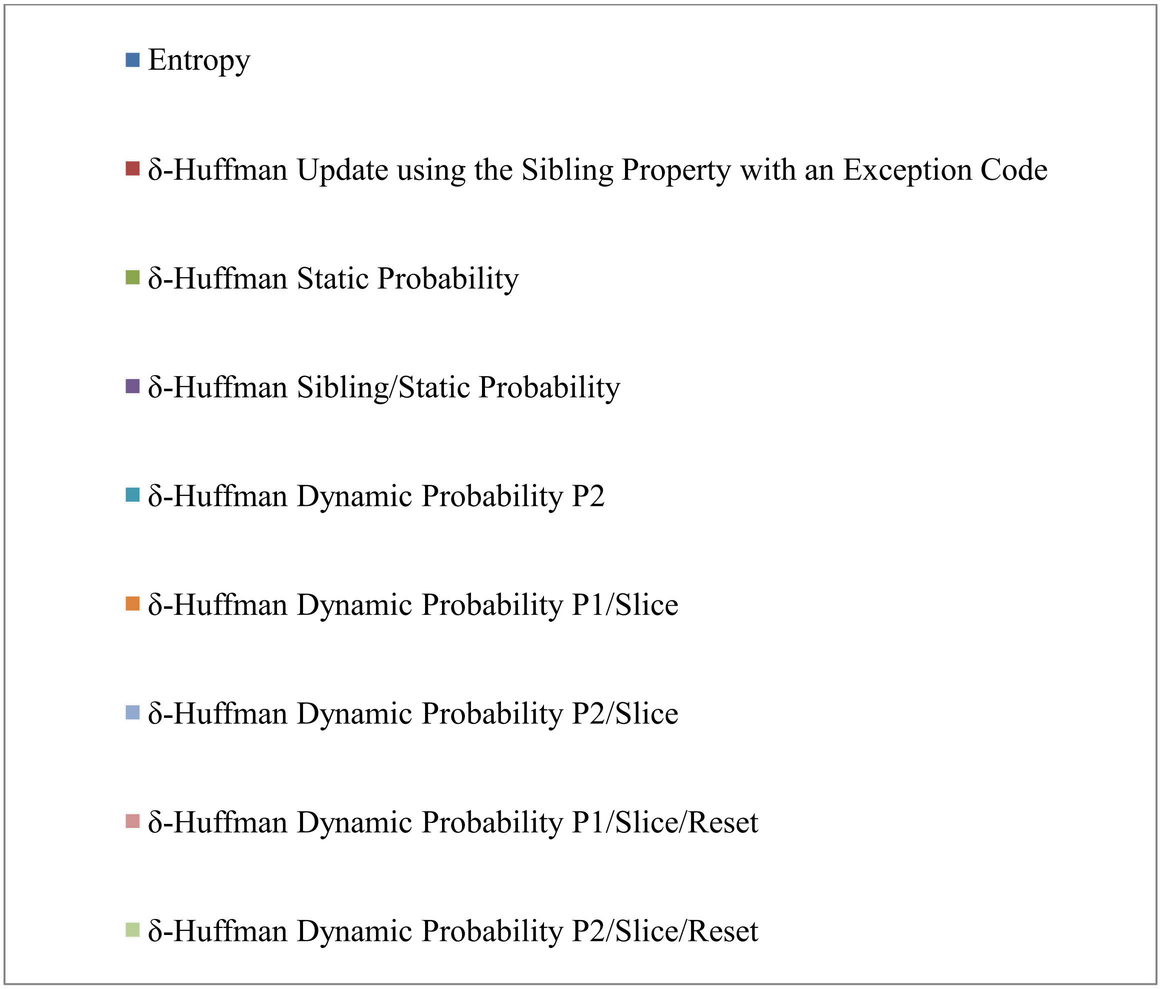


**Figure 5.32: Legend for Figure 5.33**

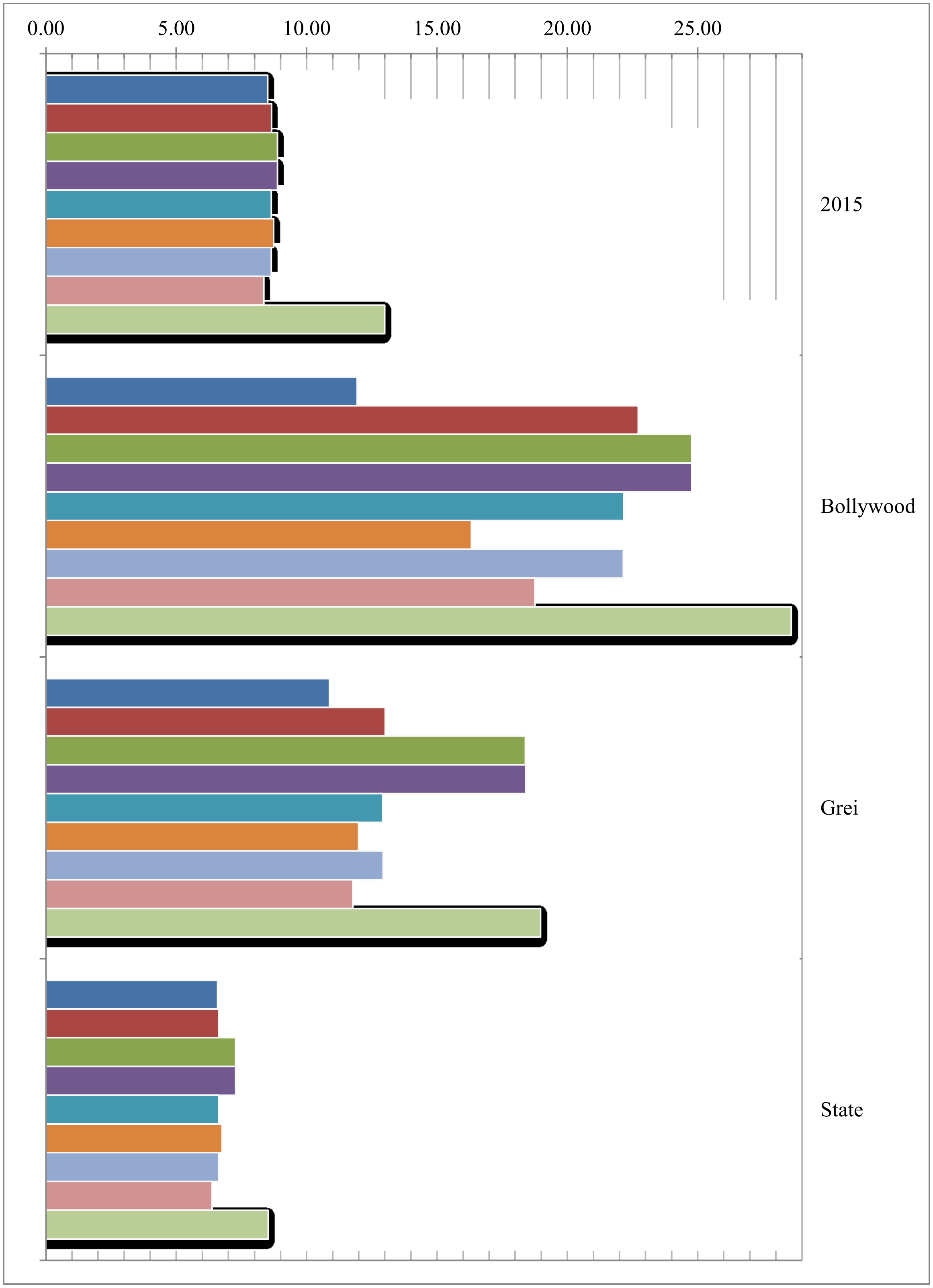


**Figure 5.33: δ-Huffman variants bit rate values vs. Entropy for real-world data set from Wikipedia without Rousei**

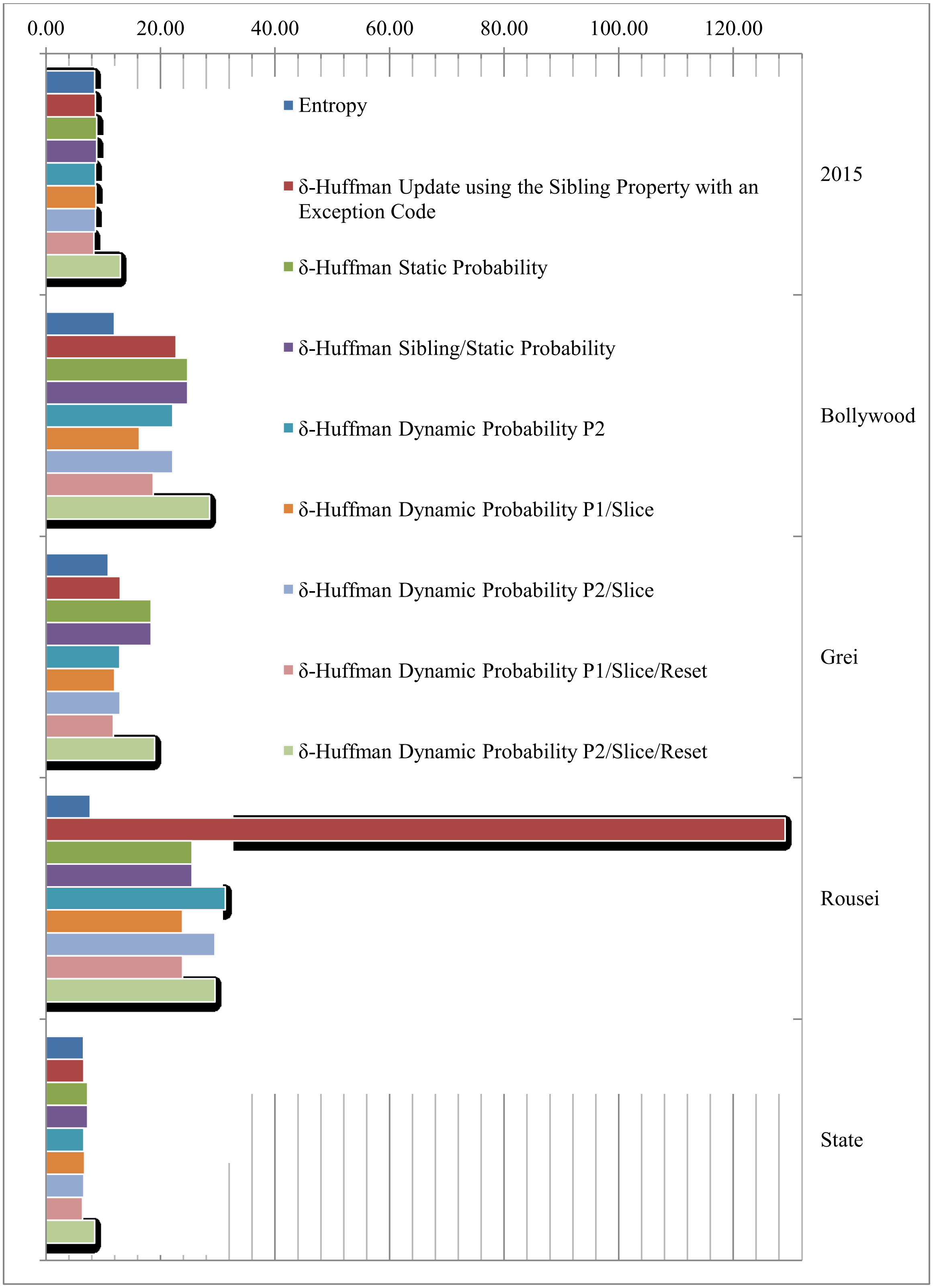


**Figure 5.34: δ-Huffman variants bit rate values vs. Entropy for real-world data set from Wikipedia with Rousei**

The last figure for section 5.13.1, figure 5.36 shows the bit rate value results of all $\delta$-Huffman variants experiments with the benchmark data set Silesia and the entropy of the data set. To display all the data in one figure, the data label value for each file groups in figure 5.36 indicates the entropy value for that file.

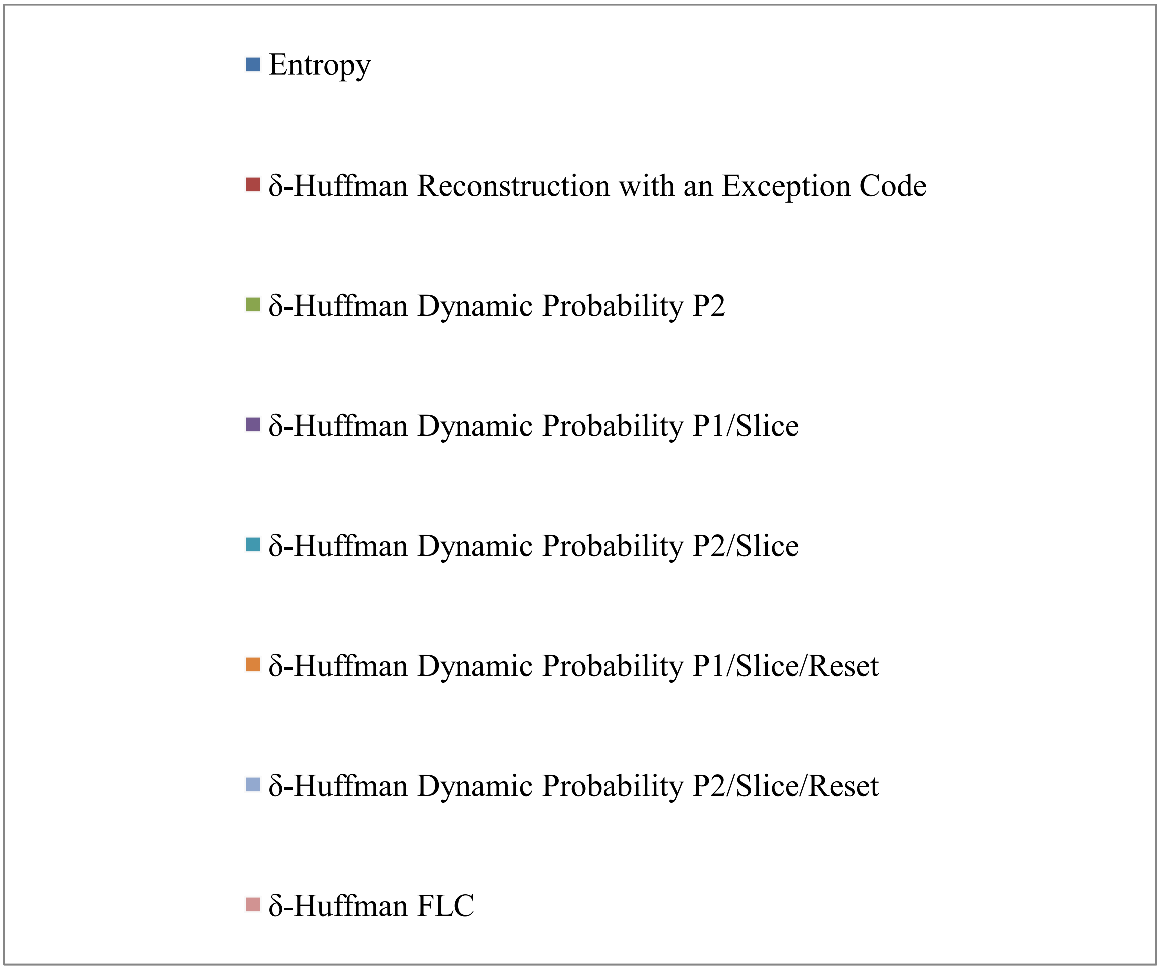


**Figure 5.35: Legend for Figure 5.36**

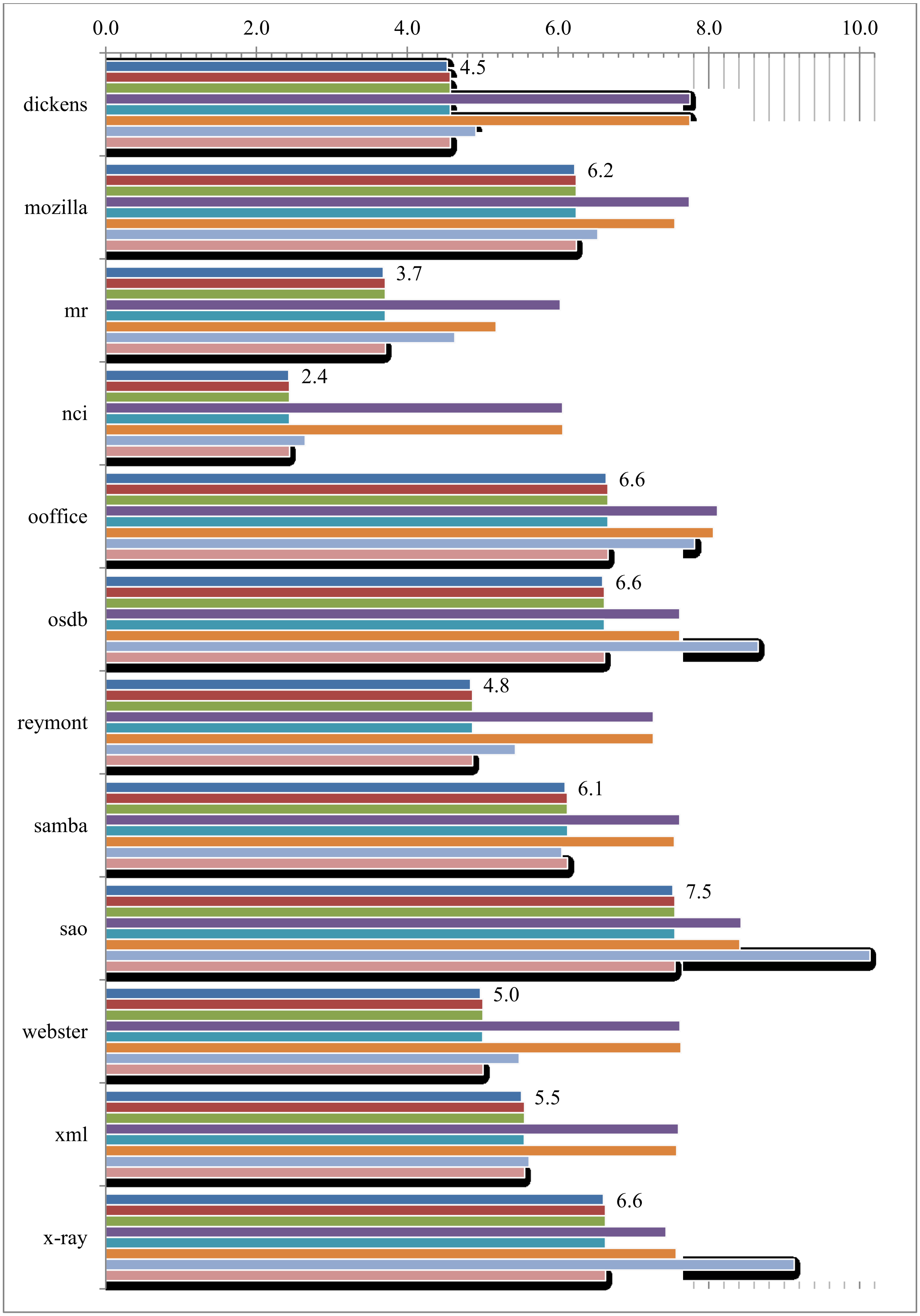


**Figure 5.36: δ-Huffman variants bit rate values vs. Entropy for each file in the data set Silesia**

**5.13.2 Results of data sets (bytes).** Figure 5.37 compares the bit rate values of the $\delta$-Huffman variants experiments with the synthetic data set handled as bytes.

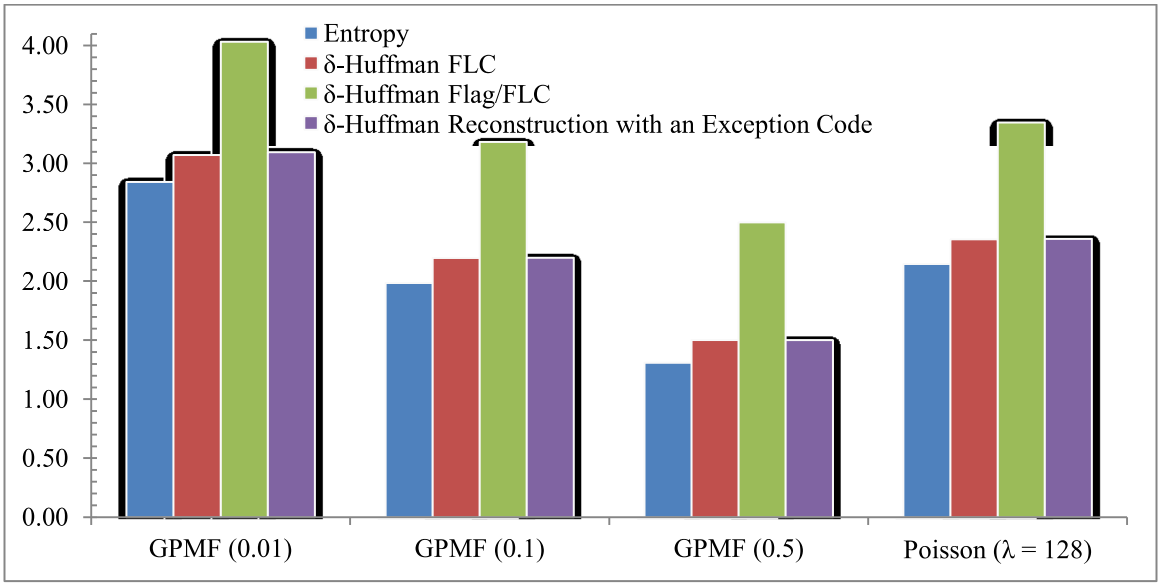


**Figure 5.37: δ-Huffman variants bit rate values (bytes) vs. Entropy for the synthetic data set**

Figure 5.38 compares the bit rate values of the $\delta$-Huffman (FLC) variant experiments with the real-world data set Wikipedia handled as bytes.

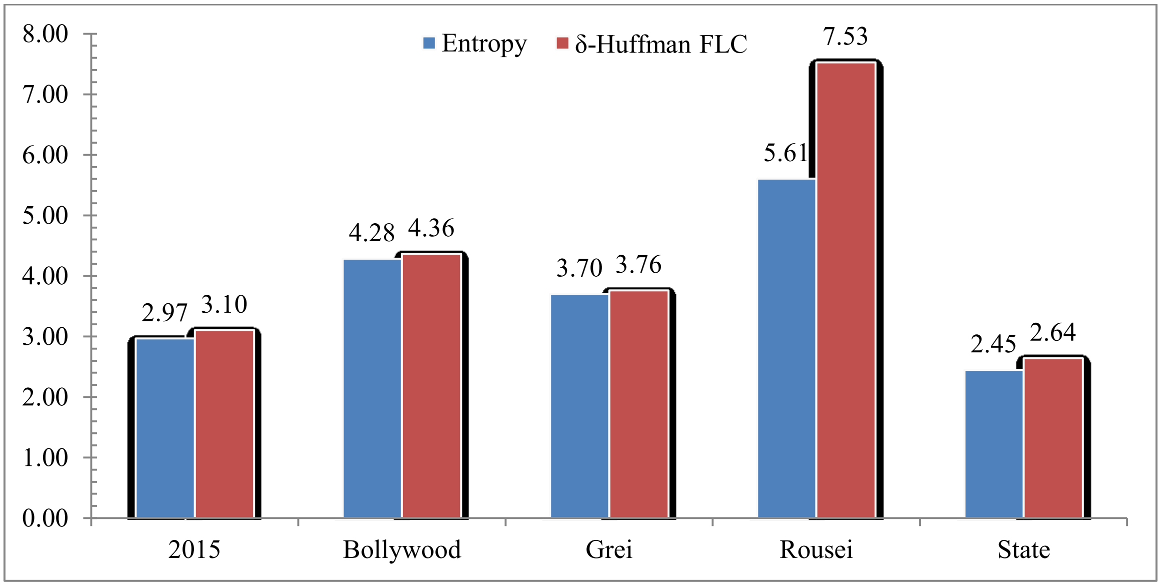


**Figure 5.38: δ-Huffman (FLC) bit rate values (bytes) vs. Entropy for real-world data set from Wikipedia**

# 6. RESULTS EVALUATION

This chapter evaluates the results and discusses the variations of the compression algorithms. This chapter focuses on global observations for the experiments and to show the differences between some of the $\delta$-Huffman bit rate value results; this chapter expands the bit rate value results beyond the second decimal place. To further explain the bit rate results of the $\delta$-Huffman (Dynamic Probability P1/Slice/Reset) algorithm that is seen in Chapter 5 in experiment 10b for the real-world data sets from Wikipedia, a brief explanation is given first.

From section 2.1.2 in Chapter 2: entropy is the theoretical lower bound on the communication bit rate over a noiseless communication channel. The entropy $(H)$ of a source $(X)$ with alphabet $A_x = \{a_1 \dots a_n\}$ and probabilities $(p_1 \dots p_n)$ is given by $H(X) = -\sum_{i=1}^{n} p_i \log_2(p_i)$ [1]. This formula is used to generate the entropy values for all data sets used in the experiments. All $\delta$-Huffman experiments met this criterion, with the exception of the $\delta$-Huffman (Dynamic Probability P1/Slice/Reset) algorithm where it performed slightly better than the entropy values for Wikipedia data set file “2015” by 0.150-bit and file “State” by 0.2092 bit.

These slightly better values were further evaluated. A plausible explanation for this is the dynamic methods of the $\delta$-Huffman (Dynamic Probability P1/Slice/Reset) algorithm and the characteristics of these two files. From section 2.7.10 in Chapter 2, the algorithm uses the first 16 input data values to learn about the probability characteristic of the files prior to generating the first Huffman tree that is used by the encoder and decoder. The algorithm uses the dynamic probability formula $(P1) = (1 - \alpha)^{(n-1)} \ * \alpha$ to generate values for symbols in the Huffman tree, which is updated multiple times during the first 128 iterations but not between every iteration. Then the symbols in the Huffman tree are zeroed out and updated only at designated iterations identified as a slice $(L)$ by the algorithm. For the characteristics of the two files, both are from Wikipedia data sets, which mostly lend themselves to following a geometric distribution function. The two files are the largest data files in the Wikipedia data sets, giving the algorithm multiple chances to have slice segments where the code in the Huffman tree are utilized efficiently and the data from the files are small values and highly concentrated among those small values. All this leads to the slightly better values in the experiment.

Global observations follow:

**Synthetic Data Evaluation**

1. The bit rate value results for the top three $\delta$-Huffman experiments for the synthetic data set GPMF (0.01) are shown in table 6.1.

**Table 6.1: δ-Huffman (Probability Assumption) vs. Difference in Entropy**

| **δ-Huffman (Probability Assumption)** | **Difference from the entropy of GMPF (0.01)** |
|---|---|
| δ-Huffman (Dynamic Probability P1/Slice) | 0.402 |
| δ-Huffman (Dynamic Probability P1/Slice/Reset) | 0.405 |
| δ-Huffman (Static Probability) | 0.406 |

From table 6.1 it can be noted that the $\delta$-Huffman (Dynamic Probability P1/Slice) is the closest to the entropy of the GMPF (0.01) data set.

2. The bit rate value results for the top three $\delta$-Huffman experiments for the synthetic data set GPMF (0.1) are shown in table 6.2.

**Table 6.2: δ-Huffman (Probability Assumption) vs. Difference in Entropy**

| **δ-Huffman (Probability Assumption)** | **Difference from the entropy of GMPF (0.1)** |
|---|---|
| δ-Huffman (Dynamic Probability P1/Slice) | 0.068 |
| δ-Huffman (Dynamic Probability P1/Slice/Reset) | 0.072 |
| δ-Huffman (Sibling/Static Probability) | 0.075 |

From table 6.2 it can be noted that the $\delta$-Huffman (Dynamic Probability P1/Slice) is the closest to the entropy of the GMPF (0.1) data set.

3. The bit rate value results for the top three $\delta$-Huffman experiments for the synthetic data set GPMF (0.5) are shown in table 6.3.

**Table 6.3: δ-Huffman (Probability Assumption) vs. Difference in Entropy**

| **δ-Huffman (Probability Assumption)** | **Difference from the entropy of GMPF (0.5)** |
|---|---|
| δ-Huffman (Dynamic Probability P1/Slice) | 0.00625 |
| δ-Huffman (Dynamic Probability P1/Slice/Reset) | 0.00635 |
| δ-Huffman (Dynamic Probability P2/Slice) | 0.00775 |

From table 6.3 it can be noted that the $\delta$-Huffman (Dynamic Probability P1/Slice) is the closest to the entropy of the GMPF (0.5) data set.

4. The bit rate value results for the top three $\delta$-Huffman experiments for the synthetic data set Poisson (λ = 128) are shown in table 6.4.

**Table 6.4: δ-Huffman (Probability Assumption) vs. Difference in Entropy**

| **δ-Huffman (Probability Assumption)** | **Difference from the entropy of Poisson (λ = 128)** |
|---|---|
| δ-Huffman (Dynamic Probability P2/Slice) | 0.0888 |
| δ-Huffman (Reconstruction with an Exception Code) | 0.0970 |
| δ-Huffman (Dynamic Probability P2) | 0.0972 |

From table 6.4 it can be noted that the $\delta$-Huffman (Dynamic Probability P2/Slice) is the closest to the entropy of the Poisson (λ = 128) data set.

Overall, based on the average of the bit rate error for each $\delta$-Huffman variant experiment with the synthetic data set; the best $\delta$-Huffman variants to evaluate the synthetic data set are the $\delta$-Huffman (Dynamic Probability P1/Slice) and the $\delta$-Huffman (Dynamic Probability P2/Slice).

5. Figure 6.1 compares the bit rate values of ten $\delta$-Huffman variants experiments with the synthetic data set to the bit rate values of [15].

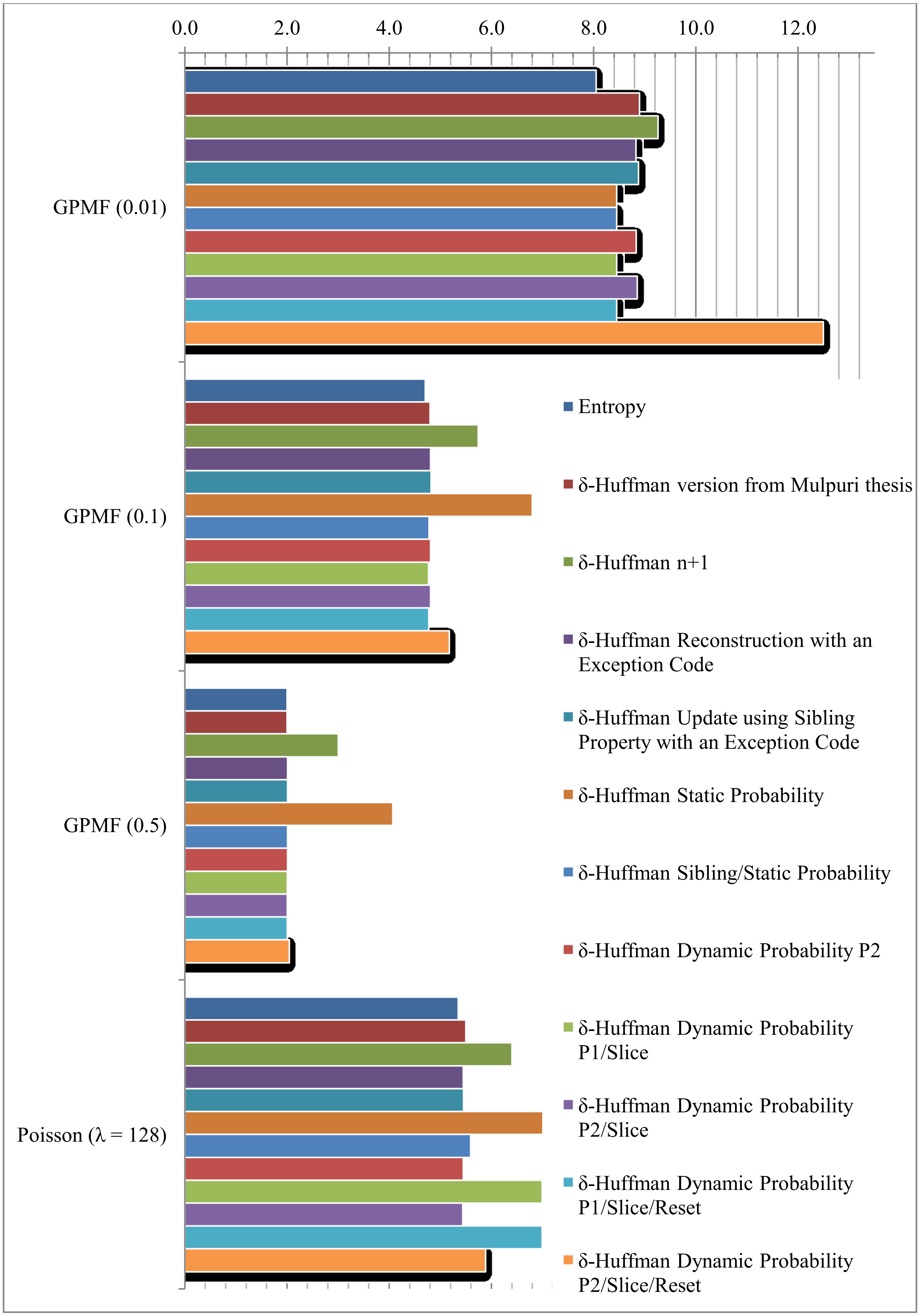


**Figure 6.1: δ-Huffman variants and [15] bit rate values vs. Entropy for the synthetic data set**

From figure 6.1 it can be noted that [15] generates an 8.9-bit rate value for GPMF (0.01). $\delta$-Huffman variants that generate lower bit rate values than [15]:

- $\delta$-Huffman (Dynamic Probability P1/Slice) with an 8.4-bit rate value.
- $\delta$-Huffman (Static Probability), $\delta$-Huffman (Sibling/Static Probability), and $\delta$-Huffman (Dynamic Probability P1/Slice/Reset) with an 8.5-bit rate value.
- $\delta$-Huffman (Reconstruction with an Exception Code) and $\delta$-Huffman (Dynamic Probability P2) with bit an 8.8-bit rate value.

the same bit rate value as [15]:

- $\delta$-Huffman (Update using Sibling Property with an Exception Code) and $\delta$-Huffman (Dynamic Probability P2/Slice).

and higher bit rate values than [15]:

- $\delta$-Huffman (n+1) with a 9.3-bit rate value.
- $\delta$-Huffman (Dynamic Probability P2/Slice/Reset) with a 12.5-bit rate value.

For GPMF (0.1), all $\delta$-Huffman variants, with the exception of the $\delta$-Huffman (Dynamic Probability P2/Slice/Reset) variant with a 5.2-bit rate value, $\delta$-Huffman (n+1) variant with a 5.7-bit rate value, and the $\delta$-Huffman (Static Probability) variant with a 6.8-bit rate value, generate the same 4.8-bit rate value as [15].

For GPMF (0.5), all $\delta$-Huffman variants, with the exception of the $\delta$-Huffman (n+1) variant with a 3.0-bit rate value and the $\delta$-Huffman (Static Probability) variant with a 4.1-bit rate value, generate the same 2.0-bit rate value as [15].

From figure 6.1 it can be noted that [15] generates a 5.5-bit rate value for Poisson ($\lambda$ = 128). $\delta$-Huffman variants that generate lower bit rate values than [15]:

- $\delta$-Huffman (Reconstruction with an Exception Code), $\delta$-Huffman (Dynamic Probability P2), and $\delta$-Huffman (Dynamic Probability P2/Slice) with a 5.4-bit rate value.

the same bit rate value as [15]:

- $\delta$-Huffman (Update using Sibling Property with an Exception Code).

and higher bit rate values than [15]:

- $\delta$-Huffman (Sibling/Static Probability) with a 5.6-bit rate value.
- $\delta$-Huffman (Dynamic Probability P2/Slice/Reset) with a 5.9-bit rate value.
- $\delta$-Huffman (n+1) with a 6.4-bit rate value.
- $\delta$-Huffman (Static Probability), $\delta$-Huffman (Dynamic Probability P1/Slice), and $\delta$-Huffman (Dynamic Probability P1/Slice/Reset) with a 7.0-bit rate value.

Overall, based on the average of the bit rate for each $\delta$-Huffman variant and [15]; $\delta$-Huffman (Sibling/Static Probability), $\delta$-Huffman (Reconstruction with an Exception Code), $\delta$-Huffman (Dynamic Probability P2), and $\delta$-Huffman (Dynamic Probability P2/Slice) have bit rate values that are the closest to [15].

However, when taking the average bit rate of [15] and comparing them to the entropy of the synthetic data set, it does not outperform the $\delta$-Huffman (Sibling/Static Probability), $\delta$-Huffman (Reconstruction with an Exception Code), $\delta$-Huffman (Dynamic Probability P2), $\delta$-Huffman (Update using Sibling Property with an Exception Code), and $\delta$-Huffman (Dynamic Probability P2/Slice).

The final note for synthetic data evaluation is that the $\delta$-Huffman (Dynamic Probability P2/Slice) is the only $\delta$-Huffman variant to perform favorably in both the experiments for the synthetic data set and in the comparison with [15].

**Real-world Data Evaluation**

6. The bit rate value results for the top three $\delta$-Huffman experiments for the real-world data set Wikipedia (2015) are shown in table 6.5.

**Table 6.5: δ-Huffman (Probability Assumption) vs. Difference in Entropy**

| **δ-Huffman (Probability Assumption)** | **Difference from the entropy of Wikipedia (2015)** |
|---|---|
| δ-Huffman (Dynamic Probability P1/Slice/Reset) | -0.150 |
| δ-Huffman (Dynamic Probability P2) | 0.141 |
| δ-Huffman (Dynamic Probability P2/Slice) | 0.142 |

From table 6.5 it can be noted that the $\delta$-Huffman (Dynamic Probability P1/Slice/Reset) is the closest to the entropy of the Wikipedia (2015) data set.

7. The bit rate value results for the top three $\delta$-Huffman experiments for the real-world data set Wikipedia (Bollywood) are shown in table 6.6.

**Table 6.6: δ-Huffman (Probability Assumption) vs. Difference in Entropy**

| **δ-Huffman (Probability Assumption)** | **Difference from the entropy of Wikipedia (Bollywood)** |
|---|---|
| δ-Huffman (Dynamic Probability P1/Slice) | 4.373 |
| δ-Huffman (Dynamic Probability P1/Slice/Reset) | 6.814 |
| δ-Huffman (Dynamic Probability P2/Slice) | 10.210 |

From table 6.6 it can be noted that the $\delta$-Huffman (Dynamic Probability P1/Slice) is the closest to the entropy of the Wikipedia (Bollywood) data set.

8. The bit rate value results for the top three $\delta$-Huffman experiments for the real-world data set Wikipedia (Grei) are shown in table 6.7.

| **Table 6.7: δ-Huffman (Probability Assumption) vs. Difference in Entropy** | |
|---|---|
| **δ-Huffman (Probability Assumption)** | **Difference from the entropy of Wikipedia (Grei)** |
| δ-Huffman (Dynamic Probability P1/Slice/Reset) | 0.894 |
| δ-Huffman (Dynamic Probability P1/Slice) | 1.117 |
| δ-Huffman (Dynamic Probability P2) | 2.039 |

From table 6.7 it can be noted that the $\delta$-Huffman (Dynamic Probability P1/Slice/Reset) is the closest to the entropy of the Wikipedia (Grei) data set.

9. The bit rate value results for the top three $\delta$-Huffman experiments for the real-world data set Wikipedia (Rousei) are shown in table 6.8.

| **Table 6.8: δ-Huffman (Probability Assumption) vs. Difference in Entropy** | |
|---|---|
| **δ-Huffman (Probability Assumption)** | **Difference from the entropy of Wikipedia (Rousei)** |
| δ-Huffman (Dynamic Probability P1/Slice) | 16.16040 |
| δ-Huffman (Dynamic Probability P1/Slice/Reset) | 16.16040 |
| δ-Huffman (Static Probability) | 17.79070 |

From table 6.8 it can be noted that the $\delta$-Huffman (Dynamic Probability P1/Slice) is the closest to the entropy of the Wikipedia (Rousei) data set.

10. The bit rate value results for the top three $\delta$-Huffman experiments for the real-world data set Wikipedia (State) are shown in table 6.9.

**Table 6.9: δ-Huffman (Probability Assumption) vs. Difference in Entropy**

| **δ-Huffman (Probability Assumption)** | **Difference from the entropy of Wikipedia (State)** |
|---|---|
| δ-Huffman (Dynamic Probability P1/Slice/Reset) | -0.2092 |
| δ-Huffman (Dynamic Probability P2) | 0.0436 |
| δ-Huffman (Dynamic Probability P2/Slice) | 0.0438 |

From table 6.9 it can be noted that the $\delta$-Huffman (Dynamic Probability P1/Slice/Reset) is the closest to the entropy of the Wikipedia (State) data set.

Overall, based on the average of the bit rate error for each $\delta$-Huffman variant experiment with the real-world data set from Wikipedia; the best $\delta$-Huffman variants to evaluate the real-world data set from Wikipedia are the $\delta$-Huffman (Dynamic Probability P1/Slice) and the $\delta$-Huffman (Dynamic Probability P1/Slice/Reset).

11. Figure 6.2 compares the bit rate values of eight $\delta$-Huffman variants experiments with the real-world data set Wikipedia to the bit rate values of [15].

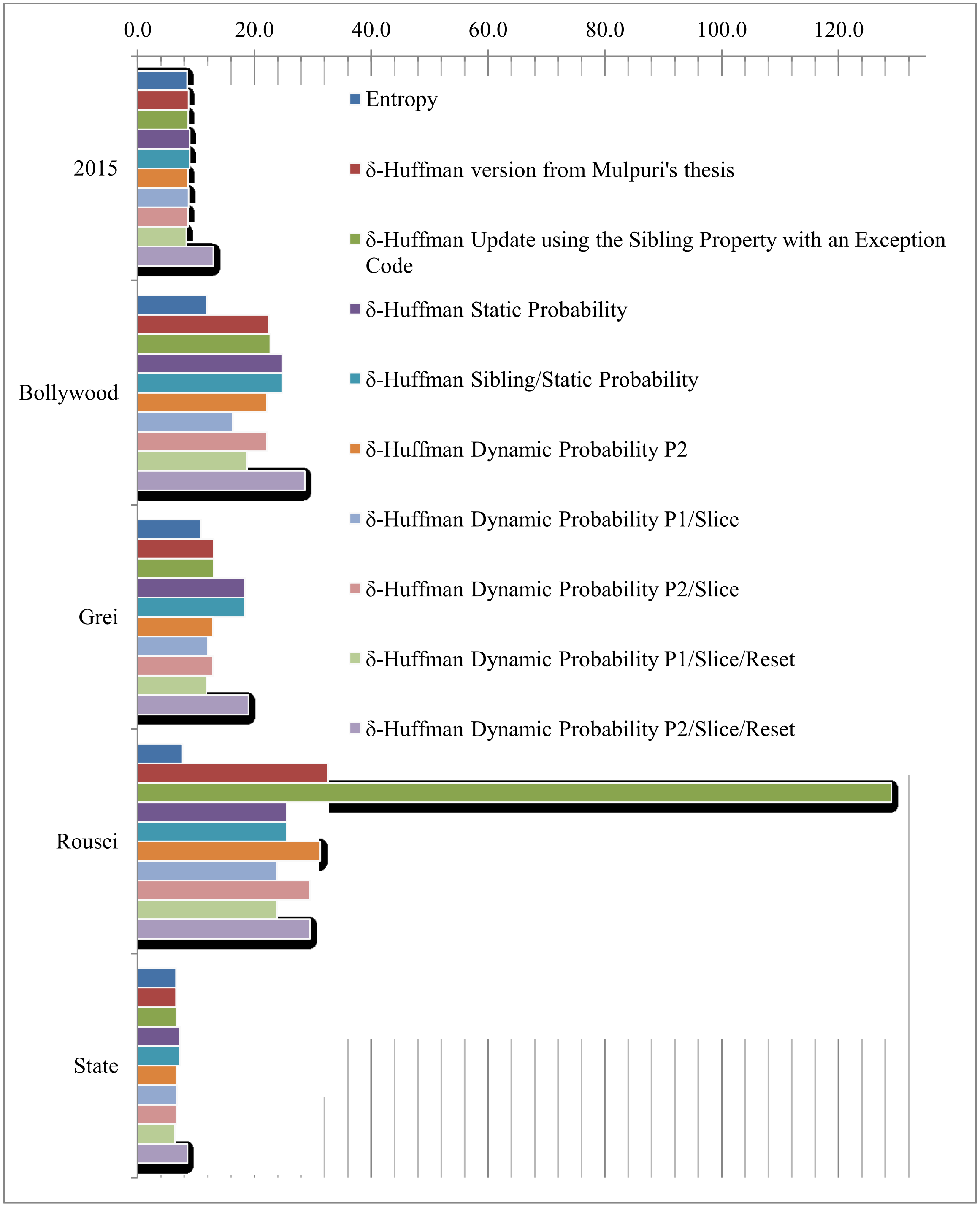


**Figure 6.2: δ-Huffman variants and [15] bit rate values vs. Entropy for Wikipedia Data**

From figure 6.2 it can be noted that [15] generates an 8.7-bit rate value for Wikipedia (2015). $\delta$-Huffman variants that generate lower bit rate values than [15]:

- $\delta$-Huffman (Dynamic Probability P1/Slice/Reset) with an 8.4-bit rate value.

- $\delta$-Huffman (Dynamic Probability P2) and $\delta$-Huffman (Dynamic Probability P2/Slice)

the same bit rate value as [15]:

- $\delta$-Huffman (Update using the Sibling Property with an Exception Code) and $\delta$-Huffman (Dynamic Probability P1/Slice).

and higher bit rate values than [15]:

- $\delta$-Huffman (Static Probability) and $\delta$-Huffman (Sibling/Static Probability) with an 8.9-bit rate value.
- $\delta$-Huffman (Dynamic Probability P2/Slice/Reset) with a 13.0-bit rate value.

From figure 6.2 it can be noted that [15] generates a 22.5-bit rate value for Wikipedia (Bollywood). $\delta$-Huffman variants that generate lower bit rate values than [15]:

- $\delta$-Huffman (Dynamic Probability P1/Slice) with a 16.3-bit rate value.
- $\delta$-Huffman (Dynamic Probability P1/Slice/Reset) with an 18.8-bit rate value.
- $\delta$-Huffman (Dynamic Probability P2) and $\delta$-Huffman (Dynamic Probability P2/Slice) with a 22.2-bit rate value.

and higher bit rate values than [15]:

- $\delta$-Huffman (Update using the Sibling Property with an Exception Code) with a 22.7-bit rate value.
- $\delta$-Huffman (Static Probability and Sibling/Static Probability) with a 24.8-bit rate value.
- $\delta$-Huffman (Dynamic Probability P2/Slice/Reset) with a 28.6-bit rate value.

From figure 6.2 it can be noted that [15] generates a 13.0-bit rate value for Wikipedia (Grei). $\delta$-Huffman variants that generate lower bit rate values than [15]:

- $\delta$-Huffman (Dynamic Probability P1/Slice/Reset) with an 11.8-bit rate value.
- $\delta$-Huffman (Dynamic Probability P1/Slice) with a 12.0-bit rate value.
- $\delta$-Huffman (Dynamic Probability P2) and $\delta$-Huffman (Dynamic Probability P2/Slice) with a 12.9-bit rate value.

the same bit rate value as [15]:

- $\delta$-Huffman (Update using the Sibling Property with an Exception Code).

and higher bit rate values than [15]:

- $\delta$-Huffman (Static Probability) and $\delta$-Huffman (Sibling/Static Probability) with an 18.4-bit rate value.
- $\delta$-Huffman (Dynamic Probability P2/Slice/Reset) with a 19.0-bit rate value.

For Wikipedia (Rousei), all $\delta$-Huffman variants, with the exception of the $\delta$-Huffman (Update using the Sibling Property with an Exception Code) variant with a 129.1-bit rate value, generate lower bit rate values than the 32.6-bit rate value by [15].

From figure 6.2 it can be noted that [15] generates a 6.6-bit rate value for Wikipedia (State). $\delta$-Huffman variants that generate lower bit rate values than [15]:

- $\delta$-Huffman (Dynamic Probability P1/Slice/Reset) with a 6.4-bit rate value.

the same bit rate value as [15]:

- $\delta$-Huffman (Update using the Sibling Property with an Exception Code), $\delta$-Huffman (Dynamic Probability P2), and $\delta$-Huffman (Dynamic Probability P2/Slice).

and higher bit rate values than [15]:

- $\delta$-Huffman (Dynamic Probability P1/Slice) with a 6.8-bit rate value.

- $\delta$-Huffman (Static Probability) and $\delta$-Huffman (Sibling/Static Probability) with a 7.3-bit rate value.
- $\delta$-Huffman (Dynamic Probability P2/Slice/Reset) with an 8.5-bit rate value.

Overall, based on the average of the bit rate for each $\delta$-Huffman variant and [15]; $\delta$-Huffman (Sibling/Static Probability), $\delta$-Huffman (Static Probability), $\delta$-Huffman (Dynamic Probability P2), $\delta$-Huffman (Dynamic Probability P2/Slice) have bit rate values that are the closest to [15].

However, when taking the average bit rate of [15] and comparing them to the entropy of the real-world data set Wikipedia, it does not outperform the $\delta$-Huffman (Dynamic Probability P2), $\delta$-Huffman (Dynamic Probability P1/Slice), $\delta$-Huffman (Dynamic Probability P2/Slice) and $\delta$-Huffman (Dynamic Probability P1/Slice/Reset).

The final note for real-world data evaluation is that the $\delta$-Huffman (Dynamic Probability P1/Slice) and $\delta$-Huffman (Dynamic Probability P1/Slice/Reset) are the $\delta$-Huffman variants to perform favorably in the experiments for the real-world data set Wikipedia but the $\delta$-Huffman (Dynamic Probability P2/Slice) is the $\delta$-Huffman variant that uses the slice method in its algorithm to perform favorably in the comparison with [15].

**Benchmark Data Silesia Evaluation**

To simplify the global evaluation of the benchmark data set Silesia, files in the Silesia data set that have the same top three $\delta$-Huffman variants are grouped together.

12. The bit rate value results for the top three $\delta$-Huffman experiments for the Silesia files dickens, mozilla, mr, nci, ooffice, osdb, reymont, sao, and x-ray are shown in table 6.10.

**Table 6.10: δ-Huffman (Probability Assumption)**

| **δ-Huffman (Probability Assumption)** |
|---|
| δ-Huffman (FLC) |
| δ-Huffman (Reconstruction with an Exception Code) |
| δ-Huffman (Dynamic Probability P2) |

From table 6.10 it can be noted that the $\delta$-Huffman (FLC) is the closest to the entropy of the Silesia files dickens, mozilla, mr, nci, ooffice, osdb, reymont, sao, and x-ray.

13. The bit rate value results for the top three $\delta$-Huffman experiments for the Silesia file samba are shown in table 6.11.

**Table 6.11: δ-Huffman (Probability Assumption)**

| **δ-Huffman (Probability Assumption)** |
|---|
| δ-Huffman (Dynamic Probability P2/Slice/Reset) |
| δ-Huffman (FLC) |
| δ-Huffman (Dynamic Probability P2) |

From table 6.11 it can be noted that the $\delta$-Huffman (Dynamic Probability P2/Slice/Reset) is the closest to the entropy of the Silesia file samba.

14. The bit rate value results for the top three $\delta$-Huffman experiments for the Silesia files webster and xml are shown in table 6.12.

**Table 6.12: δ-Huffman (Probability Assumption)**

| **δ-Huffman (Probability Assumption)** |
|---|
| δ-Huffman (Dynamic Probability P2/Slice) |
| δ-Huffman (FLC) |
| δ-Huffman (Reconstruction with an Exception Code) |

From table 6.12 it can be noted that the $\delta$-Huffman (Dynamic Probability P2/Slice) is the closest to the entropy of the Silesia files webster and xml.

For comparison, figure 6.3 shows the bit rate results of all $\delta$-Huffman variants experiments with the benchmark data set Silesia and the entropy of the data set. The data label value for each file groups in figure 6.3 indicates the entropy value for that file.

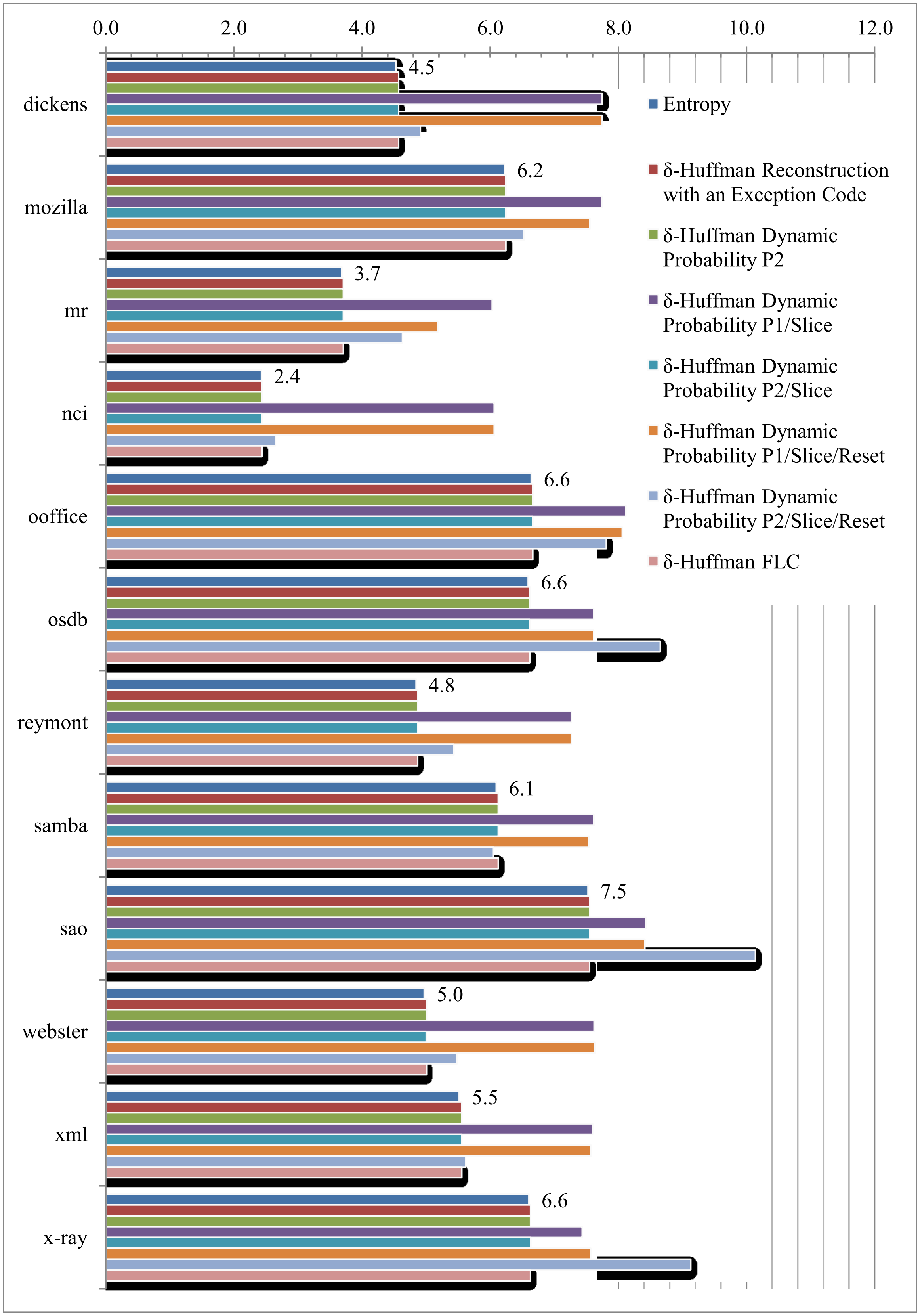


**Figure 6.3: δ-Huffman variants bit rate values vs. Entropy for each file in the data set Silesia**

Overall, the best algorithms to evaluate the benchmark data set Silesia are the $\delta$-Huffman (FLC) and the $\delta$-Huffman (Dynamic Probability P2).

The final note for benchmark data Silesia evaluation is that the $\delta$-Huffman algorithm performs better in terms of the bit rate values if prior knowledge of the type of data from the data source is known. However, given the results of the twelve experiments, if the type of data and the probability distribution of the data from a data source are unknown at the time of $\delta$-Huffman algorithm selection, the $\delta$-Huffman (Dynamic Probability P2) would be the formula this writer would start with. The $\delta$-Huffman (Dynamic Probability P2) does not always give the best bit rate values relative to entropy, as can be seen from the experiments, but it, or a variant of it, such as the $\delta$-Huffman (Dynamic Probability P2/Slice) seems to be close in performance to the best performing $\delta$-Huffman algorithm for several files across all data sets. This makes the $\delta$-Huffman (Dynamic Probability P2) a good algorithm if the type of data and the probability distribution of the data from a data source are unknown at the time of δ-Huffman algorithm selection. It is not the best performer across all data sources, but it is not the worst performer across all data sources in this thesis.

# 7. CONCLUSION AND FUTURE RESEARCH

This thesis explores the $\delta$-Huffman algorithm through manipulation of the estimation of the source probability function. Additionally, it examines the practical and theoretical compression capabilities of integer compression methods and devises methods for efficient compression of integers regardless of their specific probability distribution function.

Work by [15] explored the possibility of combining integer compression methods with a new dynamic Huffman compression algorithm known as $\delta$-Huffman and [16] explored the ability to dynamically compress data in one pass with Variable length nibbles with Tunstall (VLNT) and Delta-Tunstall ($\delta$-T). Both of them differ from our work which focuses on the examination of different assumptions about the estimation of the source probability function.

Twelve assumptions, each of which is encoded via an algorithm variant, are made about the source probability function, with the algorithms compression ratio, rather than its computation complexity as the focus of our work.

The experiments performed use one or more of three source data sets: synthetic data, real-world data from sorted inverted index gaps from Wikipedia, and benchmark data from Silesia that attempts to represent realistic workload data. The entropy estimates of these data sets and the average bit rate values of the twelve $\delta$-Huffman algorithms offer a way to construct a comparative evolution of the performance of the $\delta$-Huffman algorithms.

From these experiments, the $\delta$-Huffman (Dynamic Probability P1/Slice), $\delta$-Huffman (Dynamic Probability P2/Slice), $\delta$-Huffman (Dynamic Probability

P1/Slice/Reset), $\delta$-Huffman (FLC), and $\delta$-Huffman (Dynamic Probability P2) are identified as the best variants of the twelve $\delta$-Huffman algorithms depending on the data sets that are being evaluated. Finally, given the results of the twelve experiments, if the type of data and the probability distribution of the data from a data source are unknown at the time of $\delta$-Huffman algorithm selection, the $\delta$-Huffman 'Dynamic Probability P2' would be the formula this writer would start with.

Finally, we expanded the set of experiments for the $\delta$-Huffman algorithm and plan on further research. There are various areas of the $\delta$-Huffman algorithm still to explore and evaluate. They include areas such as the $\delta$-Huffman algorithm's cost/effectiveness performance in terms of its compression ratio, throughput, latency, and energy consumption. Implementation cost of different variants is yet another area to be evaluated.

# LITERATURE CITED

1. K. Sayood, Introduction to Data Compression, 3rd Edition, San Francisco, CA: Morgan Kaufmann, 2006.
2. D. Salomon, “Variable-Length Codes for Data Compression,” London, Springer, 2007.
3. 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.
4. C. D. Manning, P. Raghavan and H. Schutze, Introduction to Information Retrieval, New York, NY: Cambridge University Press, 2008.
5. V. N. Anh and A. Moffat, “Index Compression using Fixed Binary Codewords,” in *Proceedings of the 15th Australaian Database Conference*, Darlinghurst, Australia, 2004.
6. Vitter J. S., “Design and analysis of dynamic Huffman codes,” J. ACM 34, 4, 825-845, 1987.
7. D. Salomon, Data Compression, 4th Edition ed., London: Springer, 2007.
8. Orlitsky A., Santhanam, N.P., Junan Zhang, “Universal compression of memoryless sources over unknown alphabets,” in *Information Theory, IEEE Transactions on*, vol.50, no.7, pp.1469-1481, 2004.
9. Merhav N., Seroussi G., Weinberger M. J., “Coding of sources with two-sided geometric distributions and unknown parameters,” in *Information Theory, IEEE Transactions on*, vol.46, no.1, pp.229-236, 2000.

10. Gallager R. and Van Voorhis D. C., “Optimal Source Codes for Geometrically Distributed Integer Alphabets,” IEEE Transactions on Information Theory, Vol. 21(2), pp. 228-230, March 1975.

11. Kato A.; Han T. S., Nagaoka H., “Huffman coding with an infinite alphabet,” in *Information Theory, IEEE Transactions on*, vol.42, no.3, pp.977-984, 1996.

12. Abrahams J., “Huffman-type codes for infinite source distributions,” in *Data Compression Conference, 1994. DCC ’94. Proceedings*, vol., no., pp.83-89, 29-31, 1994.

13. Tomohiko Uyematsu, Fumio Kanaya, “Asymptotical Optimality of Two Variations of Lempel-Ziv Codes for Sources with Countably Infinite Alphabet,” IEICE trans. on Fundamentals of Electronics, Communications and Computer Sciences Vol. E89-A No.10 pp.2459-2465, 2006.

14. Yang En-hui, Jia Yunwei, “Universal Lossless Coding of Sources with Large and Unbounded Alphabets,” ISIT 2000, Sorrento, Italy, June 25-30, 2000.

15. Mulpuri, Naga Sai Gowtham, “Dynamic Unbounded Integer Compression,” Texas State University, San Marcos, Texas, 2016.

16. Hyatt, Christopher Rice, “Adaptive Single-Pass Compression of Unbounded Integers,” Texas State University, San Marcos, Texas, 2017.

17. S. Deorowicz, “Silesia Compression Corpus,” Silesia University, [Online]. Available: http://sun.aei.polsl.pl/~sdeor/index.php?page=silesia. [Accessed 1 10 2017].

18. D. Tamir, “Elias Omega Coding”, *ACM ICPC Contest* Asia-Phuket 2009/2010.