Lossless Data Compression on Unbounded Integers

Professor Dan Tamir, Alvin Xu, Patrick Peng, Samuel Tian

Abstract

The field of data compression and encoding has evolved into an ever-growing and ever-important topic, with the principles of reducing storage necessary becoming critical in an increasingly data-driven world. In this paper, we will explore the efficiency of existing encoding schemes for unbounded integer and symbol compression, including Huffman and Shannon-Fano-Elias. We will also develop new encoding schemes and analyze their compression efficiency in comparison to existing and other new encoding schemes.

1 Introduction

The primary purpose of data compression is to reduce the amount of storage necessary to transmit information. We are investigating methods for lossless data compression on the set of unbounded integers and on the set of characters, as well as methods that combine the two. We are primarily concerned with compressing data from the standard, fixed 32 bits storage for all integers and the fixed 8 bits storage for all characters to a smaller number of bits. Numerous encoding schemes for these purposes already exist, with each presenting its own advantages and with proven optimality in certain probability distributions and certain situations. However, much research is still conducted in pursuit of new, efficient methods of data compression.

This paper is primarily concerned with the lossless compression of integers assuming noiseless transmission. The compression of integers is a topic that has been extensively investigated, and there are several previously developed algorithms that deal with this issue, including Huffman coding, Elias γ, δ, and ω, Shannon-Fano-Elias, and Arithmetic coding. However, there is less research regarding the compression of unbounded integers, and many of the aforementioned algorithms require information on the upper-bound of the alphabet to be successful. There exists several universal and asymptotically optimal codes that deal with unbounded integers, primarily Elias-δ, ω, and δ-Huffman, which is a combination of Elias-δ and Adaptive Huffman coding.

In this paper we present two distinct compression techniques that operate on unbounded integers, namely δ-RNS and δ-Shannon-Fano-Elias, or δ-SFE. δ-RNS performs Elias-δ on a residue number system rather than the typical decimal number system, which is motivated by the ability of RNS to represent and compute large numbers relatively efficiency. δ-SFE combines Elias-δ and SFE in a manner similar to δ-Huffman, which can potentially yield interesting practical results in cryptography and extend the exponential decryption time of canonical SFE to unbounded integers. Theoretically, these two compression techniques may be proved to be universal and asymptotically optimal.

The contributions of this work are as follows: 1) Formulating an algorithm that applies canonical SFE and residue number systems to the set of unbounded integers. To the best of the present authors' knowledge, δ-SFE and δ-RNS have not been considered by accepted literature 2) Establishing the universality and asymptotic optimality of these two algorithms. 3) Extending the super-exponential time required to decrypt SFE codes to infinite alphabets.

The layout of the rest of the paper will be as follows: Section 2 introduces basic terminology and Elias-δ, SFE, and RNS. Section 3 includes a review of most related literature. Section 4 introduces the basic algorithms for δ-SFE and δ-RNS. Section 5 lists the experiments that we will perform on the aforementioned algorithms, as well as proofs of their universality and their asymptotic optimality. Section 6 includes details of several experiments with relevant data and results. Section 7 includes an analysis of the results obtained from the experiments. Section 8 concludes and addresses proposals for further research.

2 Background

In this section, we will define terms used through this paper and the algorithms used.

2.1 Definitions

Definition 2.1. A *fixed length encoding scheme* is an encoding scheme in which each of the symbols in the data set corresponds with a sequence of bits of fixed length.

Definition 2.2. A *variable length encoding scheme* is an encoding scheme in which each of the symbols in the data set corresponds with a sequence of bits of variable length, with symbols of greater frequency given codes of lower length so as to decrease the average code length.

Definition 2.3. A *uniquely decodable encoding scheme* is an encoding scheme in which the encrypted text can be unambiguously decoded back to the original plaintext.

Definition 2.4. The *entropy* is the theoretical minimum average number of bits required to compress the symbols within the data set, given by

$$H(x) = -\sum_{i=1} p_i \log_2 p_i.$$

There exists no encoding scheme that can perform better than the entropy.

Definition 2.5. An *instantaneous code* is a code in which every encoded letter or integer can be determined at the end of each codeword, rather than waiting for a later codeword that ends a pattern to decipher that earlier codeword.

Definition 2.6. A *prefix code* is a code in which no code is the prefix of another code. Satisfying this condition of a prefix code ensures that the resulting encoded message is instantaneous and uniquely decodable.

Definition 2.7. A *universal code* is a code that satisfies the condition that the ratio between the minimal codeword length for the present encoding scheme and the entropy is bounded by a constant. In other words,

$$\frac{E_P(L_\rho)}{\max(1, H(P))} \leq K_\rho$$

where E_P represents the minimum codeword length.

Definition 2.8. An *asymptotically optimal code* is a code that satisfies the condition that the ratio R_ρ between the minimal codeword length for the present encoding scheme and the entropy is bounded above by a constant, and the ratio approaches 1 as the entropy approaches infinity. In other words,

$$\frac{E_P(L_\rho)}{\max(1, H(P))} \leq R_\rho(H(P)) \leq K_\rho$$

with

$$\lim_{H \to \infty} R_\rho(H) = 1.$$

Definition 2.9. A *probability mass function (PMF)* is a function that assigns the probability of discrete events to some value.

Definition 2.10. A *prefix* for Elias-γ or Elias-δ is the bitstring prepended to the binary representation of the integer represented.

2.2 Elias-δ Coding

In Elias Delta (Elias-δ) coding for integers, we code the integer in binary, and then the binary representation of the number of bits used for that added to the front of that binary representation of the integer with the most significant bit removed, all preceded by a string of zeros of length corresponding to the length of that binary form of the number of bits subtracted by one. For example,

$$17 = 001010001.$$

To decode this, label the number of leading zeros as C. Then we decode the first 2C+1 bits in the string by binary, subtract one, and denote it by M. Then we will read the next M bits, and denote it L. Then our decoded integer is simply $2^M + L$.

This form of compression is asymptotically optimal on a set of integers with probability distribution [1]

$$\frac{1}{2n(\log_2(2n))^2}$$

.

2.3 Huffman Coding

Huffman Coding is a type of entropy coding that produces prefix codes and operates very close to the theoretical limit of lossless data compression. The algorithm for producing the prefix codes involves mapping the frequencies of the symbols in the data set onto a binary tree, with each of the leaves representing one of the symbols of the data set. The encoding of each of the symbols is represented by the path from the root to the leaf, with each left branch denoting a 0 in the path and each right branch denoting a 1 in the path. In the final tree, symbols with a higher frequency will have shorter paths and symbols with a lower frequency will have longer paths.

A simple algorithm for encoding the plaintext is as follows:

Step 1: Initialize a leaf node for every unique symbol in the data set. The node should store information regarding the frequency of the symbol. Push all of these nodes onto a priority queue.

Step 2: Pop off the two nodes from the priority queue with the least frequency. Initialize a new internal node such that its children are the two aforementioned node and the frequency is the sum of its children's frequencies. Push this internal node back onto the priority queue. Repeat this process until the priority queue only has one element.

Step 3: The last element in the priority queue will be the root of the Huffman tree. Recursively walk through this tree and assign the paths to the respective leaf nodes.

Step 4: Replace every instance of a symbol in the plaintext with its binary path derived from the Huffman tree.

Decoding the compressed text is more straightforward. Because Huffman coding is a prefix code, it is both uniquely decodable and instantaneous. Thus, simply walk through the Huffman tree using the information provided in the compressed text (i.e. walk to the left child if the next bit is 0, otherwise walk to the right child). Once a leaf node is reached, append the corresponding symbol to the decoded text and restart the process from the root node again.

Huffman encryption schemes operate very close to the entropy, with

$$H(x) \leq \sum_{i>=1} l_i p_i \leq H(x) + 1$$

where l_i denotes the average length of the code for symbol i. Huffman and Arithmetic coding have the same performance when $p_i = \frac{1}{2^i}$, but in all other cases Arithmetic outperforms Huffman because Huffman codes are restricted to integer lengths while Arithmetic codes are not. However, Huffman codes and other prefix coding schemes are still widely used because of their relative simplicity and efficient time complexity.

2.3.1 Adaptive Huffman Coding

Huffman coding is limited to situations where the probabilities of the symbols are known beforehand. Therefore, Huffman coding is not applicable to situations where the probabilities are updating in real time, such as live media feeds. This is also prominent in the case of a set of unbounded integers, as it is impossible to derive probabilities from this set.

Adaptive Huffman coding is a workaround for this problem, and the Huffman tree is built top-down rather than bottom-up [2].

There are several key differences between a Huffman tree and an Adaptive Huffman tree. First, there is an NYT (not yet transmitted) node, from which symbols that have not been transmitted yet will branch off from. Second, the nodes are numbered $k_1, k_2, \cdots, k_{2n-1}$, where n is the number of leaves in the tree. The numbering must satisfy two properties: if two nodes are children of the same node, then they must be numbered k_{2i-1} and k_{2i} for some $i \in \mathbb{Z}^+$ such that $i < n$; and if $k_a > k_b$, then $w_a >= w_b$ where w_i denotes the weight associated with node i. These properties are collectively known as the *sibling property.* Adaptive Huffman trees must satisfy the sibling property, and if this rule is broken at any point in the update process, certain nodes must be switched with each other to preserve this property. These switches can be computed in $O(\log \alpha)$, where α represents the number of symbols in the alphabet, yielding an overall time complexity of $O(l \log \alpha)$ where l represents the length of the message being transmitted. Aside from the NYT node, Adaptive Huffman trees are exactly identical to normal Huffman trees and thus provide the same average code length and performance.

2.4 δ-Huffman coding

Canonical and Adaptive Huffman are both limited to the compression of bounded integers, as neither alternative addresses the issue of transmitting unbounded integers from encoder to decoder. δ-Huffman is a compression scheme which addresses these issues of transmitting and storing unbounded integers and operates relatively efficiently on unbounded integers. Like Adaptive Huffman, δ-Huffman uses a NYT (not yet transmitted) node and the sibling property to efficiently update trees in intermediate steps. However, for each introduction of a new symbol, the δ code of the integer is prepended to the Huffman code of the symbol, which allows the encoder and decoder to know the identity of the symbol. Decoding involves maintaining a distinct tree for which each symbol in the compressed message is used to reconstruct the Huffman tree. Universality and asymptotic optimality of one of the two variations of δ-Huffman introduced in [5] have been proven.

2.5 Shannon-Fano-Elias Coding

Shannon-Fano-Elias Coding, or SFE, for letters is a precursor to the more utilized Arithmetic Coding. In Shannon-Fano-Elias Coding, we utilize the cumulative probability distribution function to decide on encoded forms for each letter of a random variable X. We will define a function

$$F(x) = \sum_{x_i < x} p(x_i) + \frac{1}{2}p(x).$$

Then our encoding for a letter will be the first $\left\lceil \log_2 \frac{1}{p(x)} \right\rceil + 1$ bits to the right of the decimal point in the binary expansion of $F(x)$. These bit encodings will be used to encode and to decode the letters.

This encoding method has an average codeword length between one to two more bits than the entropy, rendering it less utilized, as most other encoding methods have average codeword lengths less than one bit more than the entropy. However, recent developments in algorithms to make SFE more efficient may prove to increase performance by up to one bit, thereby making it competitive with these other methods.

2.6 Arithmetic Coding

Arithmetic coding is a often-utilized method that draws on SFE principles, in which we sequentially go through the word to be encoded and choose our next interval from the disjoint intervals created corresponding to the probability of the letters. For example, assume we wish to encode "car" with the probability distribution

$$X = \{"c", "a", "r"\}, p = \{\frac{1}{2}, \frac{1}{4}, \frac{1}{4}\}.$$

What this means is that for every interval we process, we will break it into disjoint intervals corresponding to these probabilities. Our first letter "p" has a probability of $\frac{1}{2}$, so our new interval is $(0, 0.5]$. The next letter "a" has a probability $\frac{1}{4}$, which corresponds to the interval $(0.5, 0.75]$ within our probability distribution. However, we must apply this to the interval we have already obtained of $(0, 0.5]$, so our new interval is $(0.25, 0.375]$. Our last letter "r" has probability $\frac{1}{4}$, so with the same process, our final interval is $(0.34375, 0.375]$. Then, we pick any number on this final interval to represent our encoded word.

To decode, we simply need to examine which of the disjoint intervals from the probability distribution our number is in at each step, and continue limiting the interval considered in this way until we reach the final coded word of known length. For example, with the above example, assume we had chosen 0.35. Then this is on the interval $(0, 0.5]$ for "c", and on that interval, the interval $(0.25, 0.375]$ for "a", and on that interval, the interval for $(0.34375, 0.375]$ for "r".

If our length of codeword is not known, we simply need to modify our probability distribution to add a symbol corresponding to the end of a sequence.

The average code length of this encoding scheme is proven to be less than 1 bit greater than the entropy, thereby making it a remarkably useful encoding scheme, even more than Huffman due to the ability to approach entropy without needing specific probability distributions.

2.7 Residue Number System

In the Residue Number System, we process each positive integer x through a set of k coprime moduli $\{m_1, m_2, ..., m_k\}$, taking $x \mod m_1$ for each m_n. The resulting set $\{r_1, r_2, ..., r_k\}$ is guaranteed to be unique for all

$$x < m_1 * m_2 * m_3 * ... * m_k.$$

To decode this, we use the Chinese Remainder Theorem, which states that the moduli m and the residues r are related to x by the following equation:

$$|x|_M = |\sum_{i=1}^{n} r_i |M_i^{-1}|_{m_i} M_i|_M,$$

where M represents the product $m_1 * m_2 * ... * m_k$, M_i represents M/m_i, and $|M_i^{-1}|_{m_i}$ represents the inverse of M_i in base m_i.

The Residue Number System provides many advantages, including fast operations for addition, multiplication, and subtraction, along with reduced power and complexity. However, it is difficult and inefficient to perform operations such as division, comparison, and square roots within the system. The primary advantages in computation that RNS provides is the ability to compute through handling the residues only, with no carrying over powers, thereby allowing a computation to be done much more efficiently as each one can be computed in parallel rather than sequentially.

2.8 Mixed Radix System

The Mixed Radix System is another method of encoding integers, similar to the Residue Number System. However, the Mixed Radix System uses weighted positions. Each position represents a unit that is a multiple of the next one, but not necessarily by the same factor. For example, our date-time system is a Mixed Radix System. To encode a positive integer x with a Mixed Radix System with radices $\{m_1, m_2, ..., m_l\}$, we can take a backwards approach, starting at the least important residue, r_k. To find this, we take $x \mod m_k$. Next, we subtract r_k from x, divide by m_k, and repeat the process with r_{k-1}.

Decoding the Mixed Radix System is much simpler than decoding the Residue Number System. Given residues $\{r_1, r_2, ..., r_k\}$ and radices $\{m_1, m_2, ..., m_k\}$ for an integer x, x can be determined by the equation

$$x = \sum_{i=1}^{k} r_i \prod_{j=i}^{k} m_j.$$

The Mixed Radix System is much faster than the Residue Number System for division and comparisons. However, it falls behind in addition, subtraction, and multiplication because of the need to carry over digits from other positions. Additionally, it has the inherent drawback that it must be generated sequentially, whereas the Residue Number System can be implemented in parallel.

3 Related Work

3.1 Adaptive SFE

Katti et al. have proposed an Adaptive SFE coding scheme which can apply SFE to a set of symbols for which the PMF is not known, a restraint that is not satisfied by canonical SFE [3]. The PMF is initialized as evenly distributed and is consistently updated for each introduction of a symbol through Laplace's rule of succession, which accurately determines the probability of a series of events for which a relationship is not known. From these probability distributions, the SFE intervals can be reconstructed with each additional symbol. Decoding Adaptive SFE follows this algorithm in reverse, reading bits to futher restrict the possible interval and determine the symbol to be output. However, Katti et al. do not address Adaptive Huffman on unbounded integers, for which there does not exist a proper initial PMF and representation of the integers in the SFE intervals. In this paper, we introduce a new schematic for addressing these concerns.

3.2 Truncated SFE

Shannon-Fano-Elias suffers from a lower bound of one whole bit greater than the entropy, which makes this encoding scheme not as popular as alternatives such as Huffman coding. Ruan and Katti have detailed an approach to improving the performance of SFE by truncating the right end of each codeword until the very last bit before it would no longer be prefix-free with the code immediately before and after [4]. Such a scheme ensures the resulting codes maintain unique decodability while vastly reducing the bits necessary to represent the codes. Although the resulting average codeword length is still not as optimal as Huffman, Ruan and Katti concede that this is still very competitive with a major benefit in application to encryption, in which the different codewords from using different permutations of the list of symbols adds an additional $n!$ layer of security.

4 Methodology

We introduce δ-SFE and δ-RNS, two new methods of compressing unbounded integers using symbol compression techniques, offering improvements over other integer compression techniques as well as unique advantages in their respective systems. Both of these techniques will utilize similar principles of utilizing δ code as a uniquely decodable representation for the unbounded integer. Combined with SFE and RNS, both of these techniques offer greater compression while further providing the advantages that SFE and RNS give by themselves in their applications for cryptosecurity and parallel processing. In the future, δ-Arithmetic and δ-MRS will also be explored. Experimentation upon these various new encoding schemes to determine are planned to be executed and will be presented in future work.

4.1 δ-Shannon-Fano-Elias

In δ-SFE, we will begin with a NYT containing all positive integers and a probability distribution containing nothing. We will utilize flags to signal whether a transmitted integer is within the NYT or in the AT. If a integer j in the NYT is encountered, we will simply add it to the AT and transmit the δ code of $j+1$, and utilize the initial 0 in this code as our flag for a new integer ($j+1$ is encoded rather than j so that the δ representation always beings with a 0). If instead j is in the AT, we will send a flag of 1 combined with the truncated SFE code of that integer with the current probability distribution. After encoding the integer j, we will update the probability distribution to reflect this j.

To decode, we read the flags and build our probability distribution. If a 0 is read, then bits are read including that 0 until a δ code for $j+1$ is discerned, and then transmitted and added to our probability distribution. However, if a 1 is read, then we will figure out all of the truncated SFE codes of integers in the current AT, sort in order of increasing length, and then assign k as the shortest length, read k bits from the flag and determine if it matches the current considered truncated SFE codes. If it matches, then we have found our decoded integer, and we can continue - otherwise, we will remove k from the list of SFE codes and reassign k to the code of shortest length to repeat the procedure until we find a matching code. Since all truncated SFE codes are prefix codes, we will find only one uniquely decodable representation for the integer under consideration.

4.2 γ-RNS

In γ-RNS, we will use the Elias-γ code to compress the numbers generated by a RNS. For the moduli of the RNS, we will utilize prime numbers to ensure that all of the moduli are relatively prime. Given an integer x, we will determine the minimum amount of moduli needed to uniquely encode x, namely, the least integer k such that $x < \Pi_{i=1}^{k} p_k$, where p is the set of prime numbers $\{2, 3, 5, ...\}$. After determining k, we determine the residues $\{r_1, r_2, r_3, ...\}$ and encode these in binary, padding each one to the bits needed for the largest residue possible for the given modulo, given by $\lceil \log_2 p_n \rceil$. Next, we prepend the γ prefix of k to the bitstring. The γ code is UD and instantaneous, and the residues all have fixed lengths, resulting in a UD and instantaneous complete code.

To decode, we read the γ code to determine the number of primes used in encoding the integer. Due to the fact that the encoding process uses a fixed set of primes starting from 2, the bit lengths for each residue are also fixed. The decoder reads in each residue with length $\lceil \log_2 p_n \rceil$, and when it reaches the amount of residues specified in the γ code, it stops and uses the CRT formula specified in section 2.5 to decode the encoded integer. We have proven γ-RNS to be universal.

4.3 δ-RNS

In δ-RNS, we will use the Elias-δ code to compress the numbers generated by a RNS. This follows the exact same algorithm as γ-RNS, except that we will use the δ prefix of k instead of its γ representation. This provides the same advantages for δ-RNS over γ-RNS as Elias-δ over Elias-γ - namely, much better compression as the input increases.

To decode, we read the δ code to determine the number of primes used in encoding the integer, in the same way as we would for *gamma*-RNS. Then, we use the CRT formula specified in section 2.5 to decode the encoded integer. We have proven δ-RNS to be universal.

5 Universality and Asymptotic Optimality

In this section we detail the proofs of universality and asymptotic optimality for applicable algorithms. There are several assumptions that we make when considering the performance of compression algorithms operating on the set of unbounded integers: (1) all integers are greater than or equal to 1, (2) the probability mass function of the source is monotonically increasing, so smaller integers are more probable than larger integers, (3) the probability assigned to discrete integers is non-zero. From [1], we know that a probability mass function that satisifes the above criteria also satisfies Wyner's inequality $\log_2 j \leq \log_2 \frac{1}{p(j)}$ for some symbol j.

5.1 Universality of Canonical δ-SFE

Theorem 5.1. *Canonical δ-SFE is universal, but not asymptotically optimal.*

Proof. For canonical δ-SFE, our coding scheme is defined as

$$\rho(j) = \begin{cases} SFE(NYT) + \delta(j) & \text{if } j \text{ has not been encountered yet} \\ SFE(j) & \text{otherwise} \end{cases}$$

so

$$|\rho(j)| = \begin{cases} \log\frac{1}{p(j)} + 1 + 1 + \log(j) + 2\log\log(j) + 1 & \text{if } j \text{ has not been encountered yet} \\ \log\frac{1}{p(j)} + 1 & \text{otherwise} \end{cases}$$

Notice that the expected length for the case where j has not been encountered is always greater than the expected length for the case where j has already been encountered. Therefore, we can say that $\rho(j)$ is maximized when the first case is true, and

$$\begin{aligned} E_p(L) &= \frac{1}{n}\sum_{j=1}^{n}\rho(j) \\ &\leq \frac{1}{n}\sum_{j=1}^{n}\left[\log\frac{1}{p(j)} + 1 + 1 + \log(j) + 2\log\log(j) + 1\right] \\ &\leq \frac{1}{n}\sum_{j=1}^{n}\left[\log\frac{1}{p(j)} + \log\frac{1}{p(j)} + 2\log(\log j + 1) + 3\right] \\ &\leq \frac{1}{n}\sum_{j=1}^{n}\left[2\log\frac{1}{p(j)} + 2(\log j + 1) + 3\right] \\ &\leq \frac{1}{n}\sum_{j=1}^{n}\left[2\log\frac{1}{p(j)} + 2\log\frac{1}{p(j)} + 5\right] \\ &= 4\cdot\frac{1}{n}\sum_{j=1}^{n}\frac{1}{p(j)} + \frac{1}{n}\sum_{j=1}^{n}5 \\ &\leq 4\cdot\sum_{j=1}^{n}\rho(j)p(j) + 5 \\ &= 4H(P) + 5 \end{aligned}$$

Thus,

$$\frac{E_p(L)}{\max\{1, H(P)\}} \leq \frac{4H(P)+5}{\max\{1, H(P)\}} \leq 9,$$

and canonical δ-SFE is universal because the code length is bounded above by a constant. □

5.2 Universality and Asymptotic Optimality of Flagged δ-SFE

Theorem 5.2. *Flagged δ-SFE is both universal and asymptotically optimal.*

Proof. For flagged δ-SFE, our coding scheme is defined as

$$\rho(j) = \begin{cases} \delta(j) & \text{if } j \text{ has not been encountered yet} \\ 1 + SFE(j) & \text{otherwise} \end{cases}$$

so

$$|\rho(j)| = \begin{cases} 1 + \log(j) + 2\log\log(j) + 1 & \text{if } j \text{ has not been encountered yet} \\ 1 + \log\frac{1}{p(j)} + 1 & \text{otherwise} \end{cases}$$

Let k be the number of occurrences of case 1; there will be $n-k$ occurrences of case 2. Let $S \subset [n]$ such that $|S| = k$ and the k elements of S correspond with the k occurrences of case 1. Therefore, our expected length is

$$\begin{aligned}
\frac{1}{n}\sum_{j=1}^{n}\rho(j) &= \frac{1}{n}\sum_{x\in S}|\delta(x)| + \frac{1}{n}\sum_{x\in[n]\setminus S}(1+|SFE(j)|) \\
&= \frac{k}{n}\cdot\frac{1}{k}\sum_{x\in S}|\delta(x)| + \frac{n-k}{n}\cdot\frac{1}{n-k}\sum_{x\in[n]\setminus S}(1+|SFE(j)|) \\
&= \frac{k}{n}[(1+H(P)+2\log(H(P)+1)] + \frac{n-k}{n}(H+3) \\
&\leq H(P) + \frac{k}{n}\cdot 2\log(H(P)+1) + 3 \\
&\leq H(P) + 2\log(H(P)+1) + 3 \\
&\leq H(P) + 2(H(P)+1) + 3 \\
&= 3H(P) + 5
\end{aligned}$$

Thus,

$$\frac{E_p(L)}{\max\{1, H(P)\}} \le \frac{3H(P) + 5}{\max\{1, H(P)\}} \le 8,$$

and flagged δ-SFE is universal because the ratio is bounded above by a constant. We also have

$$\begin{aligned}
\lim_{H\to\infty} \frac{E_p(L)}{\max\{1, H(P)\}} &= \lim_{H\to\infty} \frac{H(P) + 2\log(H(P) + 1) + 3}{H(P)} \\
&= \lim_{H\to\infty} 1 + \lim_{H\to\infty} \frac{2\log(H(P) + 1)}{H(P)} + \lim_{H\to\infty} \frac{3}{H(P)} \\
&= 1
\end{aligned}$$

so flagged δ-SFE is asymptotically optimal. □

5.3 Universality and Asymptotic Optimality of Increment δ-SFE

Theorem 5.3. *Increment δ-SFE is both universal and asymptotically optimal.*

Lemma 5.4. *The sequence $a_n = \max\left\{\frac{k}{n}\log_2\frac{n+1}{k+1} \,\middle|\, 0 \le k \le n\right\}$ converges to $\frac{1}{e}\log_2(e)$ as n approaches infinity.*

Proof. $\frac{k}{n}\log_2\frac{n+1}{k+1}$ has a maximum whenever the derivative is equal to 0. Thus, the function has a maximum when

$$\begin{aligned}
\frac{d}{dk}\frac{k}{n} \cdot \log_2 \frac{n+1}{k+1} &= 0 \\
\frac{1}{n} \cdot \log_2 \frac{n+1}{k+1} - \frac{k}{(k+1)\ln 2} &= 0 \\
k &= (n+1)e^{W(\frac{e}{n+1})-1} - 1
\end{aligned}$$

where $W(z)$ is the Lambert W function, which is the inverse of $f(z) = ze^z$. Notice that $f(0) = 0 \cdot e^0 = 0$, so $W(0) = 0$. Substituting this value for k into the sequence and solving for the limit yields

$$\lim_{n\to\infty} \frac{k}{n} \cdot \log_2 \frac{n+1}{k+1} = \lim_{n\to\infty} \left[\frac{(n+1)e^{W(\frac{e}{n+1})-1} - 1}{n} \cdot \log_2 \frac{n+1}{(n+1)e^{W(\frac{e}{n+1})-1} - 1} \right]$$

$$
\begin{aligned}
&= \left(\lim_{n\to\infty} \frac{(n+1)e^{W(\frac{e}{n+1})-1}}{n} - \lim_{n\to\infty} \frac{1}{n}\right) \cdot \lim_{n\to\infty} \left(\log_2 \frac{n+1}{(n+1)e^{W(\frac{e}{n+1})-1} - 1}\right) \\
&= \lim_{n\to\infty} \frac{n+1}{n} \cdot \lim_{n\to\infty} \left(e^{W(\frac{e}{n+1})-1}\right) \cdot \lim_{n\to\infty} \left(-\log_2 \frac{(n+1)e^{W(\frac{e}{n+1})-1} - 1}{n+1}\right) \\
&= \lim_{n\to\infty} \frac{e^{W(\frac{e}{n+1})}}{e} \cdot \left(-\log_2 \left(\lim_{n\to\infty} e^{W(\frac{e}{n+1})-1} + \lim_{n\to\infty} \frac{1}{n+1}\right)\right) \\
&= \frac{e^{W(0)}}{e} \cdot \left(-\log_2 e^{W(0)-1}\right) \\
&= \frac{1}{e} \cdot \log_2(e)
\end{aligned}
$$

Therefore, $\lim_{n\to\infty} a_n = \frac{1}{e} \cdot \log_2(e)$. □

Corollary 5.5. *a_n is bounded above, so $\exists\, \alpha \in \mathbb{R}^+$ such that $\forall k \in \mathbb{Z}^+$, $a_k \leq \alpha$.*

Proof. The proof of universality and asymptotic optimality of δ-SFE follows easily from Corollary 5.5 and the proof of universality and asymptotic optimality for flagged δ-SFE. □

5.4 Universality of δ-RNS

Theorem 5.6. *γ and δ-RNS are universal, but not asymptotically optimal.*

Proof. The γ-RNS code is composed of a sequence of residues with the number of residues - 1 in unary and a 1 as a separator prepended. Consider the sequence of residues, which have length

$$\{\lceil \log_2 p_1 \rceil, \lceil \log_2 p_2 \rceil, \lceil \log_2 p_3 \rceil, ..., \lceil \log_2 p_k \rceil\}$$

for some p_k in the set of primes $\{p_1 = 2, p_2 = 3, p_3 = 5, ...\}$. Rosser's paper provides several bounds that can be used to approximate this length. From 3.6, we get

$$\pi(x) < \frac{1.25506x}{\ln x}$$

for all x > 1, where $\pi(x)$ represents the number of primes less than or equal to x. From 4.6, we have

$$x - 2x^{1/2} < \theta(x)$$

for all $0 < x \leq 1420.9$ or $1423 \leq x \leq 10^8$, where $\theta(x)$ represents the natural logarithm of the product of primes less than or equal to x. Additionally, we have

$$.945x < \theta(x)$$

for all $x \geq 853$ from Theorem 10. To represent a given integer j, we attempt to find the smallest number of primes m such that the product of the first m primes is greater than j. We start by establishing an upper bound on the number under which all primes must be multiplied to obtain such a product. We can set up the equation as follows so that we are guaranteed a product larger than j. Given j, we want to find x such that $\ln_j = x - 2x^{1/2}$. Then, we have the following inequality:

$$x - 2x^{1/2} = \ln j < \theta(x).$$

Solving the equation for x, we get:

$$\begin{aligned} x - 2x^{1/2} &= \ln j \\ (x^{1/2} - 1)^2 &= \ln j + 1 \\ x^{1/2} - 1 &= (\ln j + 1)^{1/2} \\ x^{1/2} &= (\ln j + 1)^{1/2} + 1 \\ x &= ((\ln j + 1)^{1/2} + 1)^2 \end{aligned}$$

Now, we can plug x into

$$\pi(x) < \frac{1.25506x}{\ln x}$$

to get an upper bound on the number of primes needed to obtain the product. This gives the upper bound as

$$\frac{1.25506((\ln j + 1)^{1/2} + 1)^2}{\ln(((\ln j + 1)^{1/2} + 1)^2)}.$$

Next, we need to bound the length of the sequence of residues. We already have an upper bound on the primes, given by x. Therefore, all the primes' bit representations must be the same length or shorter, and can be bounded by $\log_2 x$. It follows that the total length of the residues can be bounded above by

$$\frac{1.25506((\ln j+1)^{1/2}+1)^2 \log_2((\ln j+1)^{1/2}+1)^2}{\ln((\ln j+1)^{1/2}+1)^2},$$

which simplifies to

$$\frac{1.25506((\ln j+1)^{1/2}+1)^2}{\ln 2}.$$

Therefore we have

$$\rho(j) < \frac{1.25506((\ln j+1)^{1/2}+1)^2}{\ln 2}.$$

Substituting into the average length equation, we get:

$$\begin{aligned}
E_p(L_{residues}) &= \sum_{j=1}^{|M|} P(j)\rho(j) \\
&< \sum_{j=1}^{|M|} P(j)\frac{1.25506((\ln j+1)^{1/2}+1)^2}{\ln 2} \\
&= \frac{1.25506}{\ln 2}\sum_{j=1}^{|M|} P(j)((\ln j+1)^{1/2}+1)^2 \\
&= \frac{1.25506}{\ln 2}\sum_{j=1}^{|M|} P(j)(\ln j+1+2(\ln j+1)^{1/2}+1) \\
&= \frac{1.25506}{\ln 2}\sum_{j=1}^{|M|} P(j)(\log_2 j \ln 2+2(\ln j+1)^{1/2}+2) \\
&\le \frac{1.25506}{\ln 2}(\ln 2\sum_{j=1}^{|M|} P(j)\log_2\frac{1}{P(j)}+2\sum_{j=1}^{|M|} P(j)(\ln j+1)^{1/2}+2) \\
&= \frac{1.25506}{\ln 2}(\ln 2H(P)+2\sum_{j=1}^{|M|} P(j)(\ln j+1)^{1/2}+2) \\
&< \frac{1.25506}{\ln 2}(\ln 2H(P)+2\sum_{j=1}^{|M|} P(j)(\ln j+1)+2) \\
&= \frac{1.25506}{\ln 2}(\ln 2H(P)+2\sum_{j=1}^{|M|} P(j)\ln j+4) \\
&= \frac{1.25506}{\ln 2}(\ln 2H(P)+2\ln 2\sum_{j=1}^{|M|} P(j)\log_2 j+4) \\
&\le \frac{1.25506}{\ln 2}(3\ln 2H(P)+4) \\
&= 3.78518H(P)+\frac{5.02024}{\ln 2}
\end{aligned}$$

Now that we have determined an upper bound for the length in bits of the residues, we must determine an upper bound for the length of the γ code that details how many residues are contained in the code. However, we only take the unary part or prefix of the γ code, which specifies the number of residues. Thus, the number of bits we need can be determined by log_2 of the upper bound on the number of primes needed to express a given j.

$$
\begin{aligned}
E_p(L_\gamma) &= \sum_{j=1}^{|M|} P(j)\rho(j) \\
&= \sum_{j=1}^{|M|} P(j)\lfloor \log_2(\frac{1.25506((\ln j + 1)^{1/2} + 1)^2}{\ln(((\ln j + 1)^{1/2} + 1)^2)})\rfloor + 1 \\
&= \sum_{j=1}^{|M|} P(j)\lfloor \log_2(\frac{1.25506(\ln j + 2(\ln j + 1)^{1/2} + 2)}{\ln(\ln j + 2(\ln j + 1)^{1/2} + 2)})\rfloor + 1 \\
&< \sum_{j=1}^{|M|} P(j)\lfloor \log_2(1.25506(\ln j + 2(\ln j + 1)^{1/2} + 2)\rfloor + 1 \\
&< \sum_{j=1}^{|M|} P(j)\lfloor \log_2(1.25506(\log_2 j \ln 2 + 2\log_2 j \ln 2 + 2 + 2))\rfloor + 1 \\
&= \sum_{j=1}^{|M|} P(j)\lfloor \log_2(3.78518 \log_2 j \ln 2 + 5.02024)\rfloor + 1 \\
&\leq \sum_{j=1}^{|M|} P(j)\lfloor (3.78518 \log_2 j \ln 2 + 5.02024)\rfloor + 1 \\
&\leq 6.02024 + \sum_{j=1}^{|M|} P(j)\lfloor (3.78518 \log_2 j \ln 2)\rfloor \\
&\leq 6.02024 + (3.78518 \ln 2)\sum_{j=1}^{|M|} P(j)\lfloor \log_2 j\rfloor \\
&\leq 6.02024 + (3.78518 \ln 2)\sum_{j=1}^{|M|} P(j)\log_2 \frac{1}{P(j)} \\
&= 6.02024 + 3.78518 \ln 2 H(P)
\end{aligned}
$$

Additionally, we can determine the upper bound of $E_p(L_\delta)$:

$$
\begin{aligned}
E_p(L_\delta) &= \sum_{j=1}^{|M|} P(j)\rho(j) \\
&= \sum_{j=1}^{|M|} P(j)2\lfloor \log_2\lfloor \log_2 j\rfloor + 1\rfloor + 1
\end{aligned}
$$

$$\leq 2\sum_{j=1}^{|M|} P(j)\log_2(\log_2 j + 1) + 1$$
$$\leq 2\sum_{j=1}^{|M|} P(j)(\log_2 j + 1) + 1$$
$$\leq 2H(P) + 3$$

Summing the two upper bounds for γ-RNS $(L_\gamma, L_{residues})$, we get

$$\frac{E_p(L)}{\max(1,H(P))} < \frac{(3.78518\ln 2)(1+\ln 2)H(P) + 3.78518\ln 2\ln 2 + 8.80542\ln 2 + 5.02024}{\max(1,H(P))}$$

so $\frac{E_p(L)}{\max(1,H(P))}$ is bounded above by a constant for all $0 < x \leq 1420.9$ or $1423 \leq x \leq 10^8$. Summing the two upper bounds for δ-RNS $(L_\delta, L_{residues})$, we get

$$\frac{E_p(L)}{\max(1,H(P))} < \frac{5.78518H(P) + \frac{5.02024}{\ln 2} + 3}{\max(1,H(P))}$$

Therefore γ-RNS, δ-RNS are universal for this range of x. Next, we need to prove this for $x \longrightarrow \infty$. Following a similar method, we solve for x, using the inequality given by Theorem 10.

$$.945x < \theta(x)$$
$$.945x = \ln j < \theta(x)$$
$$x = \frac{\ln j}{.945}$$

Plugging this back into $\pi(x) < \frac{1.25506x}{\ln x}$ and multiplying by $log_2(\frac{\ln j}{.945})$, the length of the binary representation of the upper bound on the values of the primes, we get the upper bound on the length of the residues expressed in binary as

$$E_p(L_{residues}) = \sum_{j=1}^{|M|} P(j)\frac{1.25506\frac{\ln j}{.945}}{\ln(\frac{\ln j}{.0980})} log_2(\frac{\ln j}{.945})$$
$$= \sum_{j=1}^{|M|} P(j)\frac{1.25506\ln j}{.945\ln 2}$$

$$
\begin{aligned}
&= \frac{1.25506}{0.945 \ln 2} \sum_{j=1}^{|M|} P(j) \ln j \\
&\leq \frac{1.25506}{0.945} H(P)
\end{aligned}
$$

We can obtain an upper bound on $E_p(L_\gamma)$ by taking $\log_2$ of the upper bound on the number of primes needed:

$$
\begin{aligned}
E_p(L_\gamma) &= \lfloor \sum_{j=1}^{|M|} P(j) \log_2 (\frac{1.25506 \frac{\ln j}{.945}}{\ln(\frac{\ln j}{.945})}) \rfloor + 1 \\
&\leq \sum_{j=1}^{|M|} P(j) \frac{1.25506}{.945} \ln j + 1 \\
&\leq \frac{1.25506 \ln 2}{.945} H(P) + 1
\end{aligned}
$$

Following the same method, we determine an upper bound for $E_p(L_\delta)$:

$$
\begin{aligned}
E_p(L_\delta) &= \sum_{j=1}^{|M|} P(j) 2 \lfloor \log_2 \lfloor \log_2 \frac{\ln j}{.945} \rfloor + 1 \rfloor + 1 \\
&\leq 2 \sum_{j=1}^{|M|} P(j) \frac{\ln j}{.945} + 3 \\
&\leq \frac{2 \ln 2}{.945} H(P) + 3
\end{aligned}
$$

Summing these two upper bounds on average lengths for γ-RNS gives an upper bound on the average length of the whole sequence for all x and therefore all j, giving:

$$
\frac{E_p(L)}{\max(1, H(P))} = \frac{\frac{1.25506 + 1.25506 \ln 2}{.945} H(P) + 1}{\max(1, H(P))}
$$

Similarly, for δ-RNS, we get:

$$
\frac{E_p(L)}{\max(1, H(P))} = \frac{\frac{2 \ln 2}{.945} H(P) + 3}{\max(1, H(P))}
$$

Therefore, as the lengths of γ-RNS and δ-RNS are bounded by constant multiples of entropy plus constants for any input, γ-RNS and δ-RNS are universal. □

6 Experiments, Results, and Evaluation

In this section we detail several experiments performed so far, list the results, and provide result evaluation. We performed numerous experiments and report on representative results.

6.1 Experimental Setup

We will evaluate the performance of our algorithms by testing the performance of these algorithms on several practical data sets. We will be evaluating the performance of canonical δ-SFE, increment δ-SFE with $k = 1$, flagged δ-SFE, and δ-RNS. Each data set will contain 10,000 integers that generated according to some preset distribution criteria. The data sets are enumerated as follows: (1a) is generated according to a geometric PMF with parameter $p = 0.5$, (1b) is generated according to a GPMF with parameter $p = 0.1$, (1c) is generated according to a GMPF with parameter $p = 0.01$, (2) is generated according a Poisson distribution with $\lambda = 128$, (3) is pseudo-randomly generated with a range from 1 to 1000, and (4) consists of Wikipedia inverted indexes for 25 pages. We compare the performance of our algorithms to Elias-δ and entropy to determine the optimality of our algorithms at a lower interval of integers.

We will also independently test the performance of δ-RNS on individual integers ranging from 1 to $2^{64} - 1$ rather than a sequence of integers because the compressed text is not dependent on the ordering of the sequence. We will compare the performance of δ-RNS to the performance of γ-RNS and δ.

6.2 Results

The data gathered from our experiments are enumerated in the following charts.

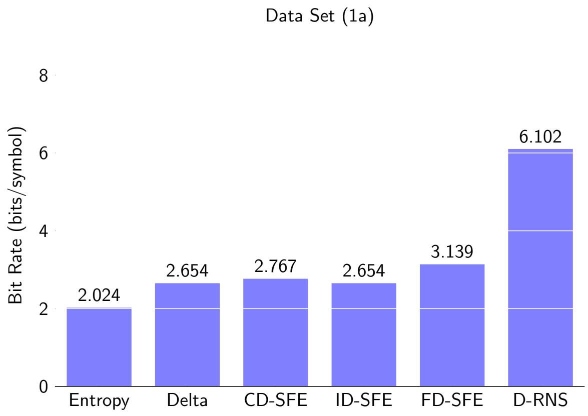

Figure 1: Data Set (1a)

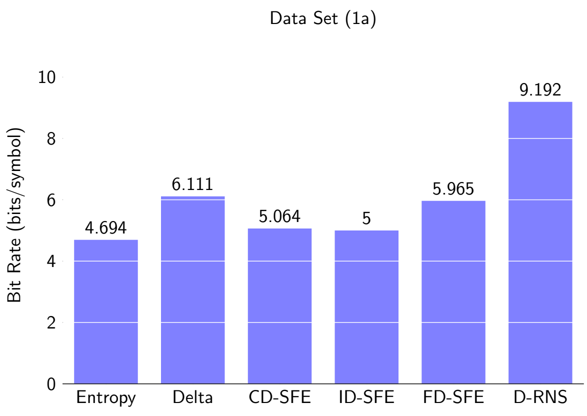

Figure 2: Data Set (1b)

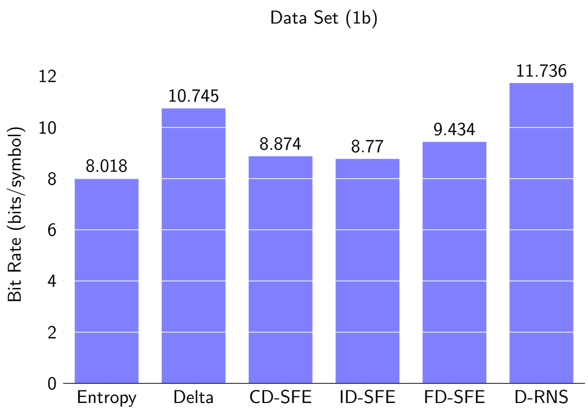

Figure 3: Data Set (1c)

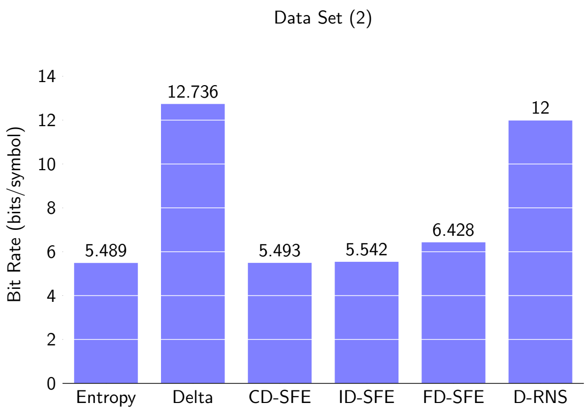

Figure 4: Data Set (2)

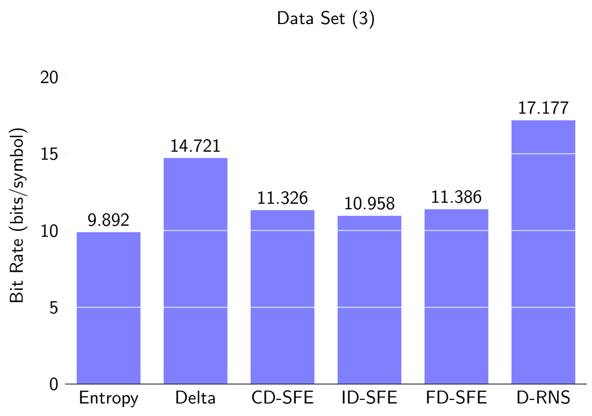

Figure 5: Data Set (3)

6.3 Evaluation

Figure 1 demonstrates the weaknesses of our algorithms, as all variants of δ-SFE only managed to perform as good or worse than Elias-δ. However, all variants of δ-SFE outper-

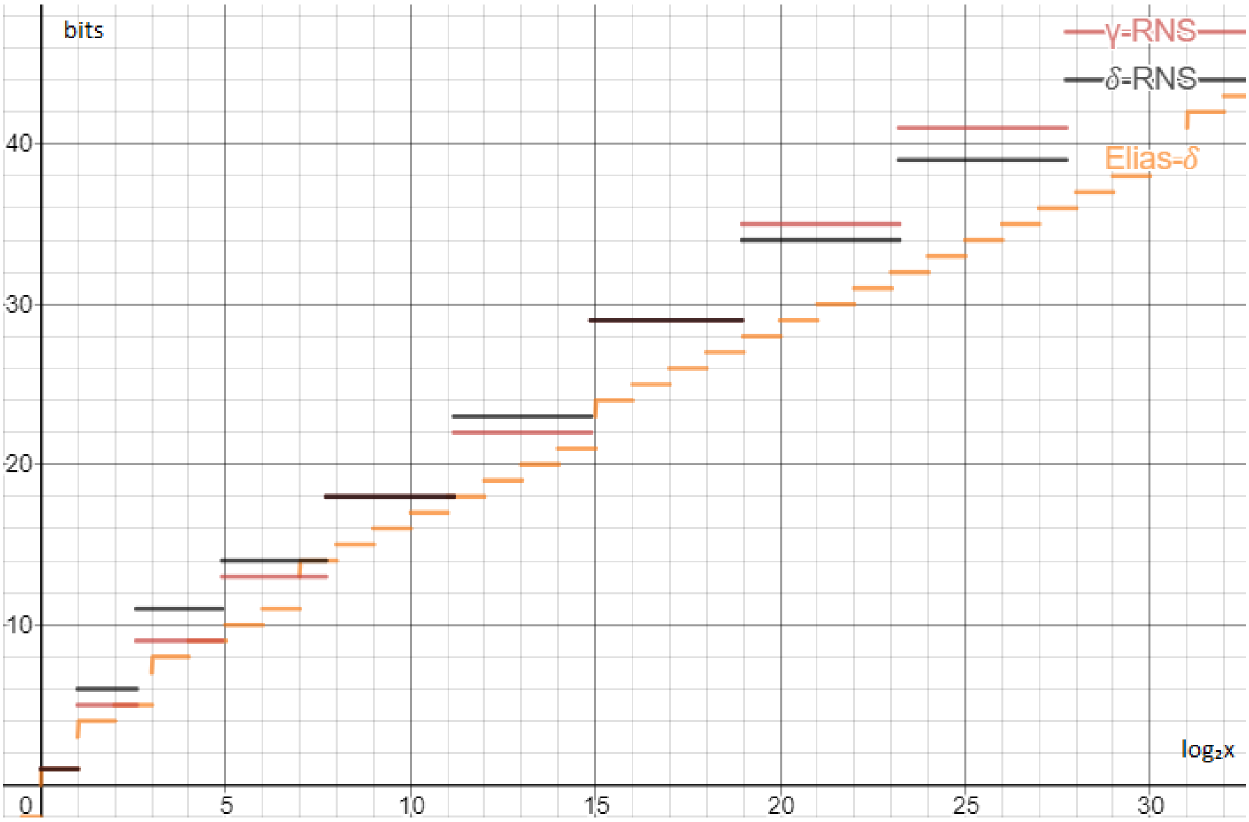

Figure 6: Elias-δ, γ-RNS, δ-RNS

form Elias-δ for all other distributions. This may be due to the inferiority of δ-SFE when it comes to compressing smaller integers because the data set had a maximum element of 15. Thus, δ-SFE should only be practically used on data sets that are composed of larger integers.

Aside from data set (1a), all variants of δ-SFE performed better than Elias-δ and operated within several bits of entropy, making these algorithms competitive with more widely used algorithms in the field of unbounded integer data compression. Both canonical δ-SFE and increment δ-SFE significantly better than flagged δ-SFE even though flagged δ-SFE is asymptotically optimal while canonical δ is not, implying that canonical δ-SFE has more practical advantages but suffers from extreme edge cases.

Notice that pseudo-randomly generated integers and Poisson distributions are not monotonically increasing distribution functions, which contradicts one of the key assumptions that we made about the source PMF. However, our algorithms still perform very close to entropy, indicating the adaptability and versatility of our algorithms and its consistent performance for all data sets.

Figure 6 demonstrates the performance of δ-RNS, γ-RNS, and Elias-δ as the input increases. As the input increases, it can be seen that δ-RNS becomes better and better than γ-RNS. Both variants still stay close to Elias-δ, though they almost never surpass it.

7 Conclusion

We have demonstrated the universality and/or asymptotic optimality for our variants of δ-SFE and δ-RNS, and we have also demonstrated the practical advantages of applying these algorithms to finite data sets. In the future, we hope to combine Elias-δ with other forms of lossless data compression such as arithmetic coding. We will also investigate applications of our algorithms to other fields of computer science, such as the potential to extend the super-exponential decryption time of canonical SFE to code sets where the alphabet can be treated as infinite and the potential for improving efficiency of arithmetic operations on large integers.

References

[1] P. Elias. "Universal codeword sets and representations of the integers". In: *IEEE Trans. Inf. Theory* 21.2 (Mar. 1975), pp. 194–203. DOI: 10.1109/TIT.1975.1055349.

[2] M. Y. Javed and A. Nadeem. "Data compression through adaptive Huffman coding schemes". In: *2000 TENCON Proceedings* (2000). DOI: 10.1109/TENCON.2000.888730.

[3] R.S. Katti and A. Ghosh. "Security using Shannon-Fano-Elias codes". In: *2009 International Symposium on Circuits and Systems* (24-27 May 2009). DOI: 10.1109/ISCAS.2009.5118356.

[4] X. Ruan and R. Katti. "Using Improved Shannon-Fano-Elias Codes for Data Encryption". In: *2006 IEEE International Symposium on Information Theory* (9-14 July 2006). DOI: 10.1109/ISIT.2006.262005.

[5] D. Tamir. "Delta-Huffman Coding of Unbounded Integers". In: *2018 Data Compression Conference* (27-30 March 2018). DOI: 10.1109/DCC.2018.00081.