Start/Stop Codes

Steven Pigeon
Université de Montréal
pigeon@iro.umontreal.ca

Abstract

We introduce the *Start/Stop* codes, a generalization of (*Start*, *Step*, *Stop*) codes [Fiala 89]. *Start/Stop* codes can be optimized to minimize average code length according to a distribution function. The optimization can take into account machine limitations to allow fast encoding or decoding. They can also be used as efficient universal codes. In this paper, we present average code length functions and optimization methods for both (*Start*, *Step*, *Stop*) and *Start/Stop* codes. We also show that on non-increasing distribution functions, *Start/Stop* codes perform nearly as well as Huffman codes.

1 Introduction

(*Start*, *Step*, *Stop*) codes were introduced as an efficient mean of encoding offsets and lengths in a dictionary-based compression scheme [Fiala 89]. They did not, however, specify how to optimize these codes nor which class of distribution they fit best. Due to their internal structure, (*Start*, *Step*, *Stop*) codes can only perform well with integers drawn from a strictly non-increasing distribution function. Futhermore, as the internal structure of the (*Start*, *Step*, *Stop*) codes is somewhat rigid, as we will explain in the next section, the same codebook can be generated for many different distributions. In this paper, we present a variation, the *Start/Stop* codes, that removes most of the rigidity of the (*Start*, *Step*, *Stop*) codes. The *Start/Stop* codes have more flexibility because codebook generation is not controlled by only one parameter, *Step*, but by as many parameters as needed. This, we will explain in detail in section 3.

In section 2 and 3, we present in detail the structure of both types of codes and of the corresponding optimization algorithms. Codebooks, from both types of codes, resulting from optimization, will be compared against Huffman codebooks generated from the same test distributions.

2 Structure and optimization of (*Start*, *Step*, *Stop*) codes

A (*Start*, *Step*, *Stop*) code is completely characterized by its three non negative integer parameters, *Start*, *Step* and *Stop*. The parameter *Stop* is of the form $Start + k_{max}\, Step$, for $k_{max} \in \mathbb{N}$. The codes are bipartite. They are composed of a prefix that is the truncated unary coding of k, and of a suffix of $Start + k\, Step$ bits that encodes the integer itself, minus the number of codes that preceed the first code of the k th code bank. The *Start* parameter specifies the starting length of the suffixes. The *Step* parameter specifies how many bits are added in each new bank, so that the suffix length grows in function of k, the bank number. The suffix length are always of the form $Start + k\, Step$. Finally, *Stop*, or rather k'_{max}, controls the maximum length of the suffixes. The parameter k'_{max} is chosen so that all the integers in the wanted range receive a code. A $(3, 2, 9)$ code (an example from [Fiala 89]) would be of the form

Code	Range
$\boxed{0}\boxed{x_2\ x_1\ x_0}$	0 – 7
$\boxed{1\ \ 0}\boxed{x_4\ x_3\ x_2\ x_1\ x_0}$	8 – 39
$\boxed{1\ \ 1\ \ 0}\boxed{x_6\ x_5\ x_4\ x_3\ x_2\ x_1\ x_0}$	40 – 167
$\boxed{1\ \ 1\ \ 1}\boxed{x_8\ x_7\ x_6\ x_5\ x_4\ x_3\ x_2\ x_1\ x_0}$	168 – 699

where the last prefix is 111 rather than 1110. Using 1110 would waste a bit since we know that there are only four code banks: after having read 3 bits to 1, we know that we are in the last code bank.

Let $l(k)$ and $h(k)$ be such that $l(0) = 0$ and

$$l(k+1) = h(k) = \frac{2^{Start+(k+1)Step} - 2^{Start}}{2^{step} - 1} \tag{1}$$

the lower and upper bounds of the k^{th} bank. In the k^{th} bank, all codes will be allocated to integers $l(k) \leqslant i < h(k)$. If $0, \ldots, N-1$ is the range of the integers to code, the parameters will be chosen so that $h(k_{max}) \geqslant N$. This ensures that all integers receive a code. The code for i is given by

$$C_{(Start,Step,Stop)}(i) = C_{\hat{\alpha}(k_{max})}(k(i)) : C_{\beta(start+k(i)Step)}(i - l(k(i)))$$

where $C_{\hat{\alpha}(k)}(i)$ is the k-truncated unary code of i, that is, the unary code of i, using upto k bits long, and $C_{\beta(n)}(i)$ is the natural binary code for $(i \mod 2^n)$ on n bits. This notation follows the notation of Elias [Elias 75]. The code bank containing i is given by

$$k(i) = \left\lfloor \frac{\lg\left(i(2^{Step} - 1) - 2^{Start}\right) - Start}{Step} - 1 \right\rfloor \tag{2}$$

a result obtained by solving $l(k) \leqslant i$ for the largest k and where $\lg x$ is shorthand for $\log_2 x$. The average code length for a random variable X and parameters *Start*, *Step*, and *Stop* is given by

$$\begin{aligned}
\bar{L}_{(Start,Step,Stop)}(X) &= \sum_{k=0}^{k_{max}} P(l(k) \leqslant X < h(k)) L_{(Start,Step,Stop)}(x \mid k) \\
&= \sum_{k=0}^{k_{max}} P(l(k) \leqslant X < h(k))(\min(k_{max}, k+1) + Start + k\ Step) \\
&= Start + 1 + (Step + 1)E[\,k\,] - P(l(k_{max}) \leqslant X < h(k_{max}))
\end{aligned} \tag{3}$$

The best distribution function for this type of code is a piecewise uniform distribution where the pieces are distributed exponentially, that is,

$$\begin{aligned}
P(X = x) &= P(X = x \mid k)P(K = k) \\
&= 2^{-(Start+(k(x)+1)Step)}\, 2^{-min(k_{max}, k(x)+1)}
\end{aligned}$$

which is straightforward to show. The information contents of the prefix bits is maximized when the k th bank is half as likely as the $k-1$ th bank, that is, banks are distributed according to a probability of $P(K=k)=2^{-min(k_{max},k+1)}$ for the k th bank. Equal suffix lengths within a bank suggest that the information contents of the bits are maximized when the numbers within a bank are drawn uniformely, giving a probability for each number in bank k of $P(X=x \mid k)=2^{-(Start+k\,Step)}$.

The optimization of the $(Start, Step, Stop)$ codes consists in finding $Start$, $Step$, and $Stop$ that both minimize average code length according to the random variable X and satisfies the constraint that $h(k_{max}) \leqslant N$ when the range of the integer to code is $0,\ldots,N-1$. If we have an analytic expression for $P(X=x)$ we can find a closed form for eq.(3), and use its derivatives relative to $Start$ and $Stop$ to find the best possible parameters that minimize eq.(3) while satisfying the constraint that all integer receive a code. If $P(X=x)$ is a non-increasing distribution function, eq.(3) is a concave function and we use any zero-finding algorithm to find the best parameters.

This is not always possible because oftentimes we do not know the parameters of the random variable X nor even its class, we can only gather observations drawn from X in an histogram. Let $CDF(x)=P(X \leqslant x)$, the cumulative distribution function. The CDF is computed in $O(N)$ from the histogram entries. Using the CDF will partially compensate for holes in the histogram that may result from unsufficient sampling. Since $Start + Step \leqslant \lceil \lg N \rceil$, one can easily consider examining all of the $O((\lg N)^2)$ possible values for $Start$ and $Step$. Checking the constraint amounts to verify if there exists a k such that $h(k) \geqslant N$, (using either eq.(1) or eq.(2)). Computing average code length with eq.(3) is done in $O(\lg N)$ steps. It suffices to use the CDF to compute $P(l \leqslant X < h)$ since $P(l \leqslant X < h) = CDF(h-1) - CDF(l-1)$. This result in an exact optimization of the $(Start, Step, Stop)$ code in $O(N + (\lg N)^3)$ steps, if the CDF is not already available, and in $O((\lg N)^3)$ otherwise.

3 Structure and Optimization of *Start/Stop* Codes

The $(Start, Step, Stop)$ codes are overly rigid because they constrain the code lengths to be of the form $\min(k_{max}, k+1) + Start + k\,Step$. Rather than starting with $Start$ bits and adding $Step$ bits to each new code bank, we will allow the number of bits that are added to be variable. Let $\{m_0, m_1, \ldots, m_{k'_{max}}\}$, $m_j \in \mathbb{Z}^*$, be the parameters of the *Start/Stop* code. The code lengths will now be of the form $\min(k'_{max}, k+1) + \sum_{j=0}^{k} m_j$, which much more flexible. For example, a *Start/Stop* code of parameters $\{0,3,2,0\}$ would look like

Code	Range
$\boxed{0}$	0
$\boxed{1\quad 0}\boxed{x_2\quad x_1\quad x_0}$	$1-8$
$\boxed{1\quad 1\quad 0}\boxed{x_4\quad x_3\quad x_2\quad x_1\quad x_0}$	$9-40$

1	1	1	x_4	x_3	x_2	x_1	x_0	41 – 72

Similarly to the (*Start*, *Step*, *Stop*) codes, we define $l'(0) = 0$ but

$$l'(k+1) = h'(k) = \sum_{i=0}^{k} 2^{\sum_{j=0}^{i} m_j}$$

The functions $l'(k)$ and $h'(k)$ are the lower and upper bounds of the k th *Start/Stop* code bank. We will also need the function $k'(i) = \max\{k \,|\, l'(k) \leqslant i\}$, the index of the code bank containing i, an integer to code. The function $k'(i)$ can be tabulated and searched. The tabulation will require $O(\lg N)$ integers to be stored, and search will be in $O(\lg \lg N)$ time. The code for i is given by

$$C_{\{m_0,\ldots,m_{k'_{max}}\}}(i) = C_{\hat{\alpha}(k'_{max})}(k'(i)) : C_{\beta(\sum_{j=0}^{k} m_j)}(i - l'(k'(i)))$$

The average length function is given by

$$\bar{L}_{\{m_0,m_1,\ldots,m_{k'_{max}}\}}(X) = \sum_{k=0}^{k'_{max}} P(l'(k) \leqslant X < h'(k)) \left(min(k'_{max}, k+1) + \sum_{j=0}^{k} m_j \right) \tag{4}$$

which doesn't simplify much. Here also, the task is to choose the parameters that minimize average code length while satisfying the constraint that $h'(k'_{max}) \geqslant N$, where $0, \ldots, N-1$ is the range of integers to encode.

3.1 Greedy optimization of *Start/Stop* codes

One possible course of action is to optimize first for m_0, then for m_1, upto $m_{k'_{max}}$ in a greedy fashion. The optimization procedure uses the CDF again, that is, $CDF(x) = P(X \leqslant x)$, which is computed in $O(N)$. Finding m_0 such that $P(X < 2^{m_0}) \approx \frac{1}{2}$ asks for $O(\lg N)$ operations since it suffice to find the closest cut to $\frac{1}{2}$ with the CDF. To find m_1, we try to minimize the difference between $P(X < 2^{m_0} + 2^{m_0+m_1} \,|\, X \geqslant 2^{m_0})$ and $\frac{1}{2}$ in the same way, and so on until the constraint $h'(k'_{max}) \geqslant N$ is satisfied. Note that the m_j may be zeroes, but this is not a problem since the range of representable numbers still grows by $2^{\sum_{l<j} m_l}$, as a new bank is added to the code. The number of cuts made by the algorithm depends on the distribution. For a distribution of the type $P(X = x) \sim 2^{-x}$, we will have to do $O(N)$ cuts, which is prohibitive! Furthermore, the *Start/Stop* code will degenerate in a unary code. Setting the maximum number of cuts to $O(\lg N)$ will keep the optimization procedure from degenerating. Since we have at most $O(\lg N)$ cuts, each being computed in at most $O(\lg N)$ steps, for a total of $O((\lg N)^2)$.

This optimization method gives good results, as we see in section 5, but because the optimality principle does not apply in this case, it may fail to discover the best parameter set.

3.2 Combinatorial optimization of *Start/Stop* codes

One may consider minimizing eq.(4) exactly by examining all possible sets of parameters that satisfy the constraint that all integers in the range $0, \ldots, N-1$ receive a code. Let $n = \lceil \lg N \rceil$

and M be the maximum number of parameters to optimize. The number of combination to examine, if we set $\sum_{j=0}^{M-1} m_j = n$, is given by

$$g(n, M) = \sum_{i=1}^{M} \binom{n+i-1}{n} = \binom{M+n}{M-1} \tag{5}$$

which is found using binomial identities. As fearful as it seems, this function does not grow as fast as one may think. Using Sterling's approximation for $n!$ we find that

$$g(n, M) \approx \frac{1}{\sqrt{2\pi}} (M-1)^{\frac{1}{2}-M} (n+1)^{-\frac{3}{2}-n} (M+n)^{\frac{1}{2}+M+n}$$

where terms nearly cancel each other out. For $g(16,4)$ we find 1140, for $g(32,4) = 7140$, etc. Search can be further pruned whenever we find a partial solution that gives a larger average code length than the best solution found so far.

If the CDF is already computed, we have an algorithm in $O(g(n,M)\,M)$. The brutal computation of eq.(4) is $O(M^3)$, but we can get it down to $O(M)$. Since we have M parameters, $k'_{max} = M-1$. We can compute $P(l'(k) \leqslant X < h'(k))(\min(k+1, M-1) + \sum_{j=0}^{k} m_j)$ from $P(l'(k-1) \leqslant X < h'(k-1))(\min(k, M-1) + \sum_{j=0}^{k-1} m_j)$ in $O(1)$. We have $l'(k) = h'(k-1)$, which requires only an assignment, this is $O(1)$. We also have that $h'(k) = l'(k) + 2^{\sum_{j=0}^{k} m_j}$; if we replace this identity by $h'(k) = l'(k) + 2^{S(k)}$, where $S(k) = S(k-1) + m_k$, $h'(k)$ can be computed in $O(1)$ from $l'(k)$. $P(a \leqslant X < b)$ itself is $O(1)$ if a and b are already computed. Finally, computing code length is $O(1)$ since we can compute the length of the k th bank by adding m_k+1 (or just m_k if $k = k'_{max} = M-1$) to the length of the $k-1$ th bank. We repeat this series of computation M times, giving $O(M)$ computation of the average code length provided the CDF is already computed and that integer arithmetic operations are $O(1)$ (which may not be true if the integers get really large).

Since this algorithm examine exhaustively all possible combinations, it is garanteed to find the best possible set of M parameters. The results are presented in section 5 where it is compared to the (*Start*, *Step*, *Stop*) codes and the *Start*/*Step* codes optimized with the greedy algorithm.

3.3 Optimization of *Start*/*Stop* codes for efficient coding and decoding

We may have to consider further constraining the optimization of a code to better suit the underlying machinery. In addition to satisfying $h'(k'_{max}) \geqslant N$, the code parameters could also satisfy a constraint of the form

$$\min(k'_{max}, k+1) + \sum_{j=0}^{k} m_j \equiv 0 \pmod{b}$$

where $b \in \mathbb{N}$ is some number of bits that is friendly to the machine, say 4 or 8. In this way, we may reduce the number of bit-oriented manipulations needed to extract (or emit) the code from (to) the compressed data. The resulting parameters can then be used to generate actual computer language code for the coding and decoding routines. For example, for a hypothetical distribution, if we set $b = 4$, we could obtain $\{3,3,3,0\}$ as parameters, giving a code structured as follows:

Code	Range
$\boxed{0\ \ x_2\ \ x_1\ \ x_0}$	0 – 7
$\boxed{1\ \ 0\ \ x_5\ \ x_4}\boxed{x_3\ \ x_2\ \ x_1\ \ x_0}$	8 – 71
$\boxed{1\ \ 1\ \ 0\ \ x_8}\boxed{x_7\ \ x_6\ \ x_5\ \ x_4}\boxed{x_3\ \ x_2\ \ x_1\ \ x_0}$	72 – 583
$\boxed{1\ \ 1\ \ 1\ \ x_8}\boxed{x_7\ \ x_6\ \ x_5\ \ x_4}\boxed{x_3\ \ x_2\ \ x_1\ \ x_0}$	584 – 1096

which is readily used for efficient coding and decoding. If we constrain $M \leqslant b$, we have can read (or write) the prefix in a single block of b bits, and therefore determine the length of the code before reading (writing) the rest in a single operation.

4 $(Start, Step, Stop)$ and $Start/Stop$ codes as universal codes

A universal code is a code that can be used for arbitrarily large integers. We assume that the integers are drawn from a distribution such that if $x < y$ then $P(X = x) \geqslant P(X = y)$, and $P(X = x) \neq 0, \forall\, x \in \mathbb{Z}^*$. A universal code must satisfy

$$\frac{\sum P(X = x) L_c(x)}{\sum P(X = x) \lg x} = K \geqslant 1$$

were K exists, and $L_c(x)$ is the length of the code for x. The closest to one K is, the more efficient the code is. The Kraft-McMillan theorems state that $K \geqslant 1$, otherwise the code would be ambiguous or undecipherable. In pratice, since the conditions on the distribution are not sufficient to completely specify it, we rather calculate the limit of the ratio $L_c(x)/\lg(x)$ to obtain the universality constant K.

We will show that both $(Start, Step, Stop)$ and $Start/Stop$ codes can be used as universal codes.

A $(Start, Step, Stop)$ code with parameters $(Start, Step, \infty)$ with $Start \geqslant 0$ and $Step > 0$ will provide a universal code. This kind of universal code can be made as efficient as wanted by controling *Step*. The code length for $n \in \mathbb{Z}^*$ is given by

$$\begin{aligned} L_{(Start,Step,\infty)}(n) &= Start + (k(n) + 1) + k(n)Step \\ &= Start + 1 + (Step + 1)k(n) \end{aligned}$$

and $k(n)$ is given by eq.(2). It follows that, for a very large n,

$$\begin{aligned} L_{(Start,Step,\infty)}(n) &= Start + 1 + (Step + 1)k(n) \\ &\approx Start + 1 + (Step + 1)\left(\frac{\lg(n(2^{Step} - 1) - 2^{Start}) - Start}{Step} - 1\right) \\ &\approx Start + 1 + (Step + 1)\left(\frac{\lg n + Step - Start}{Step} - 1\right) \\ &= Start + 1 + \frac{Step + 1}{Step}(\lg n - Start) \end{aligned}$$

and the universality constant is given by the limit

$$\lim_{x\to\infty} \frac{L_{(Start,Step,\infty)}(n)}{\lg n} = \frac{Step + 1}{Step} = 1 + \frac{1}{Step}$$

which can be made arbitrarily close to 1. To get an efficient code for the small integers, we can optimize *Start* and *Step* with the constraint that $Step > 0$. This will give us an universal code with universality constant of $(Step + 1)/Step$ that is still efficient with small integers.

We can do the same with *Start/Stop* codes. The parameters of a *Start/Stop* code are given by $\{m_0, m_1, \ldots, m_{k'_{max}}\}$, with $m_j \in \mathbb{Z}^*$, $\forall j \in \{0, \ldots, k'_{max}\}$. These parameters will allow us to efficiently code a finite number of integers, from zero upto $h'(k'_{max}) - 1$ (which may be larger than $N - 1$). Allowing an infinite parameter set, of the form $\{m_0, m_1, \ldots, m_k, m_{k+1}, m_{k+1}, \ldots\}$ with $m_{k+1} > 0$, will tranform a *Start/Stop* code into a universal code. The first $k + 1$ parameters are optimized on the useful range of numbers, and a parameter $m_{k+1} > 0$ is chosen to prolongate the code in the advent of a very large number needs to be coded.

Neglecting the first few parameters, the *Start/Stop* code of a verly large integer $n \in \mathbb{N}$ becomes essentially its base $2^{m_{k+1}}$ representation, preceeded by a prefix that encodes the length of the representation. The base $2^{m_{k+1}}$ representation will require $\log_{2^{m_{k+1}}} n$ base $2^{m_{k+1}}$ "digits". As each "digits" requires m_{k+1} bits to be represented, we have that $m_{k+1} \log_{2^{m_{k+1}}} n = \lg n$, that is, $\lg n$ bits are required. The prefix being $(\lg n)/m_{k+1}$ bits long (since there are $(\lg n)/m_{k+1}$ blocks of m_{k+1} bits in the suffix), the total code length is $O(\lg n + \lg n)/m_{k+1}) = O((1 + 1/m_{k+1}) \lg n)$ bits. This comes down to using the derivation for the $(Start, Step, Stop)$ codes setting $Step = m_{k+1}$ and $Start = \sum_{j=0}^{k} m_j$.

We see that $(Start, Step, Stop)$ and *Start/Stop* codes are strongly related, as both have auniversality constant of $1 + 1/a$, where a is the parameter controlling the universal portion of the code.

5 Results

Let the reader consider table 1. In this table, we compare the best $(Start, Step, Stop)$, *Start/Stop* and Huffman codes for a series of distributions with non-increasing mass functions.

Huffman codes are tree-structured minimal redundancy codes [Huffman 52]. Codebooks generated by Huffman's algorithm always have average code lengths that are less than one bit longer than what the entropy of the source would ask for. However, this algorithm requires that a description of the code tree be sent to the decoder. In our case where if $x < y$ then $P(X = x) \geqslant P(X = y)$ we can transmit only the shape of the tree. Would we be dealing with a distribution that is not strictly non-increasing, we would have to send information about the permutation of the integers that transforms this distribution into a non-increasing distribution. Sending only the shape of the tree need $O((\lg N)^2)$ bits in the average case, but can degenerate in $O(N)$. The number of bits needed to describe a $(Start, Step, Stop)$ code is $O(\lg N)$ since it suffices to encode *Start* and *Step*, which are both $O(\lg N)$ bits numbers. Describing a *Start/Step* codes asks for $O(\lg N + \lg k'_{max} + \lg g(n, k'_{max}))$ bits. The function $g(n, k)$, eq.(5), gives the number of possible combinations of M parameters that satisfy $\sum_j m_j = n$. An efficient code for $g(n, k)$ can be obtained by Cover's algorithm [Cover 73].

We see that *Start/Stop* codes, either optimized with the greedy algorithm or the combinatorial search algorithm consistently beat $(Start, Step, Stop)$ codes. *Start/Stop* codes also give codes whose average lengths are within a few percent of those of the optimal Huffman codes. The tables 2 and 3 give the paramaters obtained. The parameters for the $(Start, Step, Stop)$ codes are obtained by the exhaustive optimization algorithm presented in section 2. The parameters for the *Start/Stop* codes are obtained by the combinatorial optimization algorithm of section 3.2. The greedy algorithm, altough not optimal, tends to approximate the Huffman code tree and therefore generates often $O(\lg N)$ parameters. These $O(\lg N)$ parameters suffice entierely to describe the code. When we use the combinatorial optimization algorithm with a chosen M, these M parameters also suffice to describe unambiguously the code.

From table 2 one can see that $(Start, Step, Stop)$ codes, while being moderately efficient, tends to allow for more codes than required by the desired range. In the experiments, the range was $0, \ldots, 2^{16} - 1$. The parameters that minimized the average code length created ranges that were twice as large as needed. This excess in range causes about a bit per code to be wasted. *Start/Stop* codes, as can be seen from table 3, created very few excess codes.

6 Conclusion

We see that the $(Start, Step, Stop)$ codes are not as efficient as the *Start/Stop* codes, a generalization introduced in this paper. $(Start, Step, Stop)$ codes tend to give larger average code length than *Start/Stop* codes and they also allocate a lot more codes that strictly needed. This is caused by the rigidity of $(Start, Step, Stop)$ codes, in that it constrains the code length to be of the form $\min(k'_{max}, k + 1) + Start + k\ Step$. This causes several very different distributions to receive the same best parameters *Start*, *Step*, *Stop*. The *Start/Stop* codes are more flexible. One can choose a set of parameters, $\{m_0, m_1, \ldots, m_{k'_{max}}\}$, and the codes will have a length function of the form $\min(k'_{max}, k + 1) + \sum_{i=0}^{k} m_j$, which is much more flexible than the length function of $(Start, Step, Stop)$ codes. *Start/Step* codes can be optimized with or without limiting the number of parameters, and give average code lengths that are within a few percent of the equivalent Huffman codes. Special constraints can be applied to the optimization to allow for fast coding and decoding.

				Start/Stop	
Law	$\mathcal{H}(X)$	Huffman	$(Start, Step, Stop)$	greedy	$M = 6$
Zipf †	0.6670	1.2483	1.2704	1.2584	1.2675
Exp	1.4935	1.5766	1.8289	1.5766	1.6067
16 Exp ‡	5.4346	5.4637	6.2738	5.5304	5.6028
32 Exp ‡	6.4327	6.4618	6.8279	6.5282	6.6011
$\lvert\mathcal{N}(0,1)\rvert$	1.1080	1.3672	1.6420	1.3672	1.3672
$\lvert\mathcal{N}(0,10)\rvert$	4.3708	4.4028	5.6046	4.5229	4.5524
$\lvert\mathcal{N}(0,100)\rvert$	7.6894	7.7095	8.1513	7.7878	7.7968
Geometric, $p = \frac{1}{2}$	2.0000	2.0000	2.2593	2.0000	2.0854
Geometric, $p = \frac{1}{4}$	3.2381	3.2765	3.6281	3.9820	3.3824
Geometric, $p = \frac{1}{10}$	4.6824	4.7191	5.6969	4.8974	4.8048

Table 1: Comparison of the average code length for the $(Start, Step, Stop)$, $Start/Stop$ and Huffman codes for some usual distributions. † Zipf's law: $P(Z = z) \approx (z \ln(1.78N))^{-1}$. ‡ $\frac{1}{16}X \sim Exp$, $\frac{1}{32}X \sim Exp$.

$(Start, Step, Stop)$		
Law	Parameters	Range
Zipf†	$(0, 1, 16)$	0 – 131070
Exp	$(0, 1, 16)$	0 – 131070
16 Exp†	$(5, 1, 11)$	0 – 131039
32 Exp†	$(5, 1, 11)$	0 – 131039
$\lvert\mathcal{N}(0,1)\rvert$	$(0, 1, 16)$	0 – 131070
$\lvert\mathcal{N}(0,10)\rvert$	$(0, 1, 16)$	0 – 131070
$\lvert\mathcal{N}(0,100)\rvert$	$(6, 1, 16)$	0 – 131007
Geometric, $p = \frac{1}{2}$	$(0, 1, 16)$	0 – 131070
Geometric, $p = \frac{1}{4}$	$(0, 1, 16)$	0 – 131070
Geometric, $p = \frac{1}{10}$	$(0, 1, 16)$	0 – 131070

Table 2: Parameters for the $(Start, Step, Stop)$ code of table 1. Due to the rigidity of the $(Start, Step, Stop)$ codes, we end up allocating as much as twice as many codes as necessary. Notice that different random laws received the same set of parameters, an effect also due to the rigidity of the code structure. † See table 1.

Start/Stop with $M = 6$		
Law	Parameters	Range
Zipf†	$\{0, 0, 1, 2, 3, 10\}$	$0 - 65611$
Exp	$\{0, 0, 0, 0, 2, 14\}$	$0 - 65543$
16 Exp†	$\{4, 0, 0, 0, 2, 10\}$	$0 - 65663$
32 Exp†	$\{5, 0, 0, 0, 2, 9\}$	$0 - 65791$
$\lvert\mathcal{N}(0,1)\rvert$	$\{0, 0, 0, 0, 0, 16\}$	$0 - 65540$
$\lvert\mathcal{N}(0,10)\rvert$	$\{3, 0, 0, 0, 0, 13\}$	$0 - 65575$
$\lvert\mathcal{N}(0,100)\rvert$	$\{6, 0, 0, 0, 1, 9\}$	$0 - 65919$
Geometric, $p = \frac{1}{2}$	$\{0, 0, 0, 1, 1, 14\}$	$0 - 65544$
Geometric, $p = \frac{1}{4}$	$\{1, 0, 1, 0, 2, 12\}$	$0 - 65563$
Geometric, $p = \frac{1}{10}$	$\{3, 0, 0, 1, 1, 11\}$	$0 - 65607$

Table 3: Parameters of *Start/Stop* codes with $M = 6$. † See table 1.

References

[Cover 73] T. M. Cover, *Enumerative Source Coding*, IEEE Trans. Inf. Theory, v19 #1, pp. 73-77, 1973

[Elias 75] P. Elias, *Universal Codeword Sets and Representations of the Integers*, IEEE Trans. on Inform. Theory, IT-21 #2, pp. 194-203, March 1975

[Fiala 89] E.R. Fiala, D.H. Greene, *Data compression with finite windows*, CACM, v32 #4, pp. 490-505, April 1989

[Huffman 52] D. Huffman, *A Method for the construction of Minimum Redundancy Codes*, Procs. of the I.R.E., v40 #9, pp. 1098-1101, 1952