Internet Mathematics Vol. 2, No. 4: 407-429

Codes for the World Wide Web

Paolo Boldi and Sebastiano Vigna

Abstract. We introduce a new family of simple, complete instantaneous codes for positive integers, called ζ *codes*, which are suitable for integers distributed as a power law with small exponent (smaller than 2). The main motivation for the introduction of ζ codes comes from web-graph compression: if nodes are numbered according to URL lexicographical order, gaps in successor lists are distributed according to a power law with small exponent. We give estimates of the expected length of ζ codes against power-law distributions, and compare the results with analogous estimates for the more classical γ, δ and variable-length block codes.

1. Introduction

Studying web graphs[1] is often difficult due to their large size. Recently, several proposals have been published presenting different approaches to store a web graph in memory using a limited space, by exploiting both standard and *ad hoc* compression techniques.

In particular, the authors of the LINK database [Randall et al. 01] observed that if we order URLs lexicographically, successor lists (ordered so that they are monotonically increasing) tend to have small gaps between consecutive successors. Thus, instead of storing the successors explicitly, we can store their gaps using standard methods from full-text index construction, where it is customary

[1]Recall that the web graph has web pages as nodes and that y is a successor of x (said otherwise, there is an arc from x to y) whenever there is a hypertext link in page x pointing at page y. The *transpose* of the web graph, instead, has all arcs reversed (i.e., there is an arc from x to y whenever there is a hypertext link in page y pointing at page x).

to use integer variable-length instantaneous codes that assign shorter representations to smaller integers (examples of these codes may be found in Section 3).

WebGraph [Boldi and Vigna 04] is a project that follows this research path: it is a framework that provides simple methods to manage very large graphs, specially tailored around web graphs. More precisely, results of this project currently include the following:

1. A set of simple codes, called ζ *codes*, which are particularly suitable for storing web graphs (or, in general, integers with a power-law distribution in a certain exponent range). This paper defines ζ codes and estimates their expected length against power-law distributions.

2. Algorithms for compressing web graphs that exploit gap compression and differential compression (*à la* LINK [Randall et al. 01]; see also [Adler and Mitzenmacher 01] for a more in-depth theoretical analysis), intervalisation and ζ codes to provide a high compression ratio, and algorithms for accessing a compressed graph without actually decompressing it, using lazy techniques that delay the decompression until it is actually necessary. They have been described elsewhere [Boldi and Vigna 04].

3. A complete, commented implementation of the above algorithms in Java, contained in the package `it.unimi.dsi.webgraph`. Besides a clearly defined API, the package contains several classes that allow one to modify (e.g., transpose) or recompress a graph and so to experiment with various settings. It is distributed as free software under the Gnu GPL at http://webgraph.dsi.unimi.it/.

4. Data sets for very large graphs (e.g., a billion of links). These are either derived from public sources (such as WebBase [Hirai et al. 00]) or gathered with UbiCrawler [Boldi et al. 04].

During the development of WebGraph, the following question arose naturally: which is the best simple code for gaps? To choose a good way to compress gaps one has first to establish their distribution, and empirical evidence suggests that gaps of web graphs are distributed as a power law with an exponent in the range $[1.1, 1.3]$. Of course, an optimal code for such a distribution may be obtained using standard techniques such as Huffman coding, but due to the preposterous number of codewords (gaps may be as large as the number of nodes, and web graphs may have billions of nodes), this solution is not feasible: we need a *simple* code, that is, a code such that the codeword for x can be computed quickly without storing a dictionary.

Among the classical codes (see, again, Section 3) Elias' γ is suitable to encode integers distributed following power laws of exponent close to two, whereas Elias' δ is a natural choice when the exponent is close to one [Witten et al. 99]. Other powerful codes used in full-text indexing, such as Golomb codes, are not useful, as they are targeted toward exponentially decaying distributions that do not appear in practice in web graphs.

In other words, we are searching for codes whose intended behaviour lies between that of γ and of δ, that is, codes suitable for a power law with exponent between one and two. *Variable-length block codes* are sometimes used for this purpose [Randall et al. 01], but they are unsatisfactory from a theoretical and practical viewpoint because they are not optimal.

Motivated by this observation, we introduce a new class of simple codes, called ζ codes, designed for power-law distributions with small exponent (less than 1.6). Then, we compare γ, δ, ζ, and variable-length block codes by estimating their expected length against power-law distributions. The distributions we consider are heavy-tailed, so expected lengths give a very useful indication of the performance of the codes; nonetheless, the reader should not be surprised if experimenting with less than 10^9 codewords produces different results, especially with very small exponents. On the other hand, the web graph grows very quickly, so we think that estimating the expected length is a good practical measure of fitness.

Finally, we report experimental data obtained within the WebGraph framework that validate our theoretical analysis. We present comparative results about the number of bits per link necessary to code the WebBase graph [Hirai et al. 00] and a snapshot of the .uk domain of about 18,500,000 pages crawled by UbiCrawler [Boldi et al. 04]. These data are publicly available at http://webgraph.dsi.unimi.it/ and can be used to perform experiments within the WebGraph framework [Boldi and Vigna 04].

2. Power-Law Distributions

We consider the problem of determining a good instantaneous binary code for positive integers distributed as follows:[2]

$$Z_\alpha[x] = \frac{1}{\zeta(\alpha)x^\alpha}.$$

[2]In the rest of the paper, we use log to denote binary logarithms and ln to denote natural logarithms; ζ denotes the Riemann zeta function.

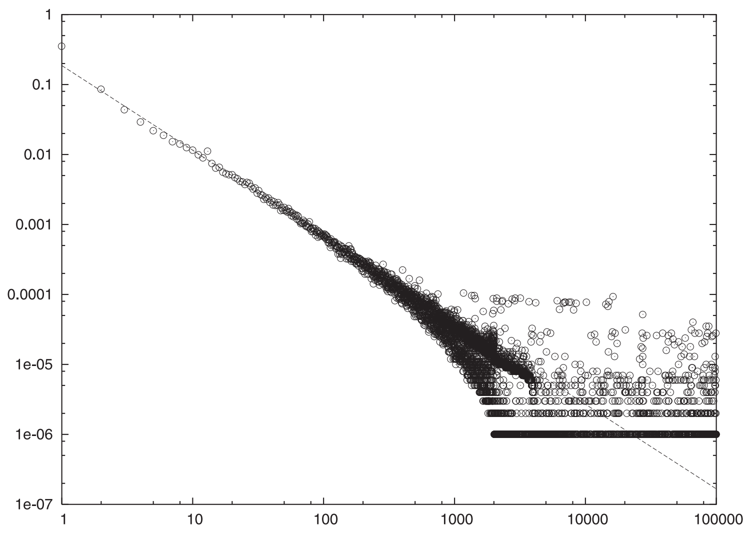

Figure 1. Distribution of gaps in a 18.5 Mpages snapshot of the .uk domain. The scale is logarithmic on both axes, and the line displays $Z_{1.21}$.

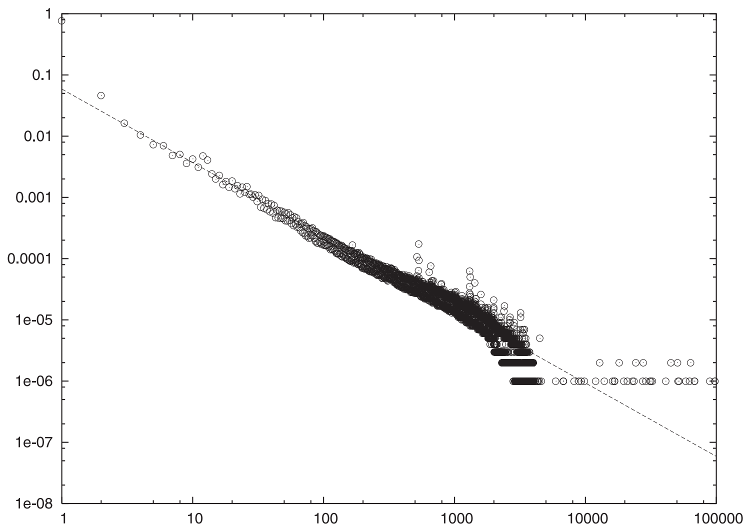

Figure 2. Distribution of gaps in the transpose of a 18.5 Mpages snapshot of the .uk domain. The scale is logarithmic on both axes, and the line displays $Z_{1.20}$ (modulo a scaling factor).

This is known as the *power-law* or *Zipf*[3] distribution with exponent $\alpha > 1$. The entropy H_α of Z_α is

$$\begin{aligned} H_\alpha &= -\sum_{x=1}^{\infty} \log\left(\frac{1}{\zeta(\alpha)x^\alpha}\right)\frac{1}{\zeta(\alpha)x^\alpha} \\ &= \frac{1}{\zeta(\alpha)}\sum_{x=1}^{\infty}(\log\zeta(\alpha) + \alpha\log x)\frac{1}{x^\alpha} \\ &= \frac{1}{\zeta(\alpha)}\left(\sum_{x=1}^{\infty}\frac{\log\zeta(\alpha)}{x^\alpha} + \alpha\sum_{x=1}^{\infty}\frac{\log x}{x^\alpha}\right) \\ &= \log\zeta(\alpha) - \frac{\alpha}{\ln 2}\frac{\zeta'(\alpha)}{\zeta(\alpha)}. \end{aligned}$$

Power-law distributions are well known in the web-graph community: many graph-theoretical parameters, such as indegrees, have been experimentally shown to have a power law distribution [Broder et al. 00].

Interestingly, the same phenomenon arises when developing techniques for compressing the web graph. Since the seminal work related to the LINK database [Randall et al. 01], it is known that by ordering lexicographically the URLs of a web graph one obtains ordered sequences of successors with small gaps between adjacent successors; understanding the distribution of such gaps is essential when choosing a good code for them.

Indeed, gathering compression statistics using WebGraph, we obtained graphs such as those of Figure 1 and Figure 2, which show gaps strictly following a power-law distribution with exponent $\alpha \approx 1.2$ for the graph, and a slightly different distribution for the gaps of the transpose. (Note that, however, in both cases the gap 1 is particularly frequent and lies above the interpolating line.) This observation is our main motivation for the pursuit of codes suitable for power-law distributions with $\alpha < 2$.

3. Introducing ζ Codes

We briefly recall the definitions of common instantaneous codes that we will use in this paper. Let x be a positive integer[4] to be coded, b its binary representation, and ℓ its length.

[3]To be precise, the Zipf distribution is an inverse polynomial function of *ranks* rather than magnitudes, but in this particular case they coincide.

[4]To simplify definitions and computations, all codes in this paper (except for minimal binary codes) represent *positive* integers only. The Java classes provided by MG4J (a Java package available at http://mg4j.dsi.unimi.it) and used within the WebGraph framework, however, provide methods that uniformly code *natural numbers*. The reader must take care of this fact when comparing the API documentation and the content of this paper.

- **Unary code**. Write $x-1$ zeroes followed by a one.
- **Levenstein's code** [Levenstein 68], **a.k.a. Elias' γ code**[5] [Elias 75]. Write ℓ in unary, followed by the last (least significant) $\ell-1$ digits of b.
- **Elias' δ code** [Elias 75]. Write ℓ in γ code, followed by the last (least significant) $\ell-1$ digits of b.
- **Variable-length nibble code**. First, pad the binary representation of $x-1$ on the left with zeroes to obtain a string whose length is a multiple of three. Then, break the string into blocks of three bits, and prefix each block with a bit, which is zero for all blocks except for the last one. This code was chosen by the authors of the LINK database [Randall et al. 01] to compress gaps.

The first few codewords for these codes are given in Table 2.

By Shannon's theory, each code has an associated *implied distribution*—the distribution (proportional to) $2^{-\ell_x}$, where ℓ_x is the length of the codeword for x. If the code is *complete* (i.e., if $\sum_x 2^{-\ell_x}=1$), then it is *optimal* with respect to its implied distribution (i.e., entropy and expected length coincide).

The implied distribution is the one for which the code is best suited, and it is in general useful to think of instantaneous integer codes in terms of their implied distributions. For instance, γ code has $\ell_x = 2\lfloor \log x\rfloor + 1$ (for integer $x>0$) and implied distribution

$$2^{-(2\lfloor \log x\rfloor+1)} \approx \frac{1}{2x^2}.$$

Thus, γ code is a good choice for power-law distributions with $\alpha \approx 2$. However, as we remarked in Section 2, the gap distributions found in web graphs have a significantly smaller exponent. On the other hand, the implied distribution for δ is

$$2^{-(2\lfloor \log\lfloor \log x+1\rfloor\rfloor+\lfloor \log x\rfloor+1)} \approx \frac{1}{2(\log(x+1))^2 x},$$

which vanishes more slowly than any Zipfian distribution, so we expect δ to be a good choice only when α is very close to one.

It is useful to notice that the exponent 2 of the implied distribution for γ code comes from two $\lfloor \log x\rfloor$ contributions: the length of the binary representation and the length of the unary code that comes before it. There is little we can do to reduce the former, but it is reasonable to try to reduce the latter.

A very well-known class of codes that uses this idea is the class of *variable-length block codes*, an obvious generalisation of variable-length nibble code in

[5]Note that this code was actually called γ' by Elias, but it is always called γ in recent literature.

	Interval				
Integer	[0,2]	[0,3]	[0,4]	[0,5]	[0,6]
0	0	00	00	00	00
1	10	01	01	01	010
2	11	10	10	100	011
3		11	110	101	100
4			111	110	101
5				111	110
6					111

Table 1. Sample minimal binary codes.

which the length of each block is an arbitrary integer k (we get back the variable-length nibble code when $k = 3$). The implied distribution of a variable-length block code with block of length k is

$$2^{-\lceil(\log x)/k\rceil(k+1)} \approx \frac{1}{x^{1+\frac{1}{k}}},$$

so the choice $k = 3$ is very reasonable,[6] since it leads to an implied power-law distribution with exponent $\alpha \approx 1.3$. This fact explains the success of variable-length nibble code in the LINK database.

Nonetheless, *variable-length block codes are not complete*, hence not optimal. Thus, we are going to introduce ζ codes, a family of complete simple codes depending on a parameter k which can be viewed as an optimal version of variable-length block codes.

First of all, let us fix a minimal binary code for intervals of integers.

Definition 3.1. The *minimal binary code* of x in the interval $[0, z-1]$ is defined as follows: let $s = \lceil \log z \rceil$; if $x < 2^s - z$, then x is coded using the xth binary word of length $s-1$ (in lexicographical order); otherwise, x is coded using the $(x - z + 2^s)$th binary word of length s.

We remark that this code is simply an optimal (Huffman) code for the uniform distribution on $[0, z-1]$. The purpose of our definition is to fix the code so that smaller integers get shorter codewords. Examples of minimal binary codes are shown in Table 1.

[6]Beware, however, that the approximation introduced by removing the floor/ceiling operators may be quite rough: this phenomenon cannot be overseen, and it will be particularly evident in the discussion that follows.

Integer	$\gamma = \zeta_1$	ζ_2	ζ_3	ζ_4	δ	nibble
1	1	10	100	1000	1	1000
2	010	110	1010	10010	0100	1001
3	011	111	1011	10011	0101	1010
4	00100	01000	1100	10100	01100	1011
5	00101	01001	1101	10101	01101	1100
6	00110	01010	1110	10110	01110	1101
7	00111	01011	1111	10111	01111	1110
8	0001000	011000	0100000	11000	00100000	1111
9	0001001	011001	0100001	11001	00100001	00011000
10	0001010	011010	0100010	11010	00100010	00011001
11	0001011	011011	0100011	11011	00100011	00011010
12	0001100	011100	0100100	11100	00100100	00011011
13	0001101	011101	0100101	11101	00100101	00011100
14	0001110	011110	0100110	11110	00100110	00011101
15	0001111	011111	0100111	11111	00100111	00011110
16	000010000	00100000	01010000	010000111	001011001	10000111

Table 2. Sample codes. Note that ζ_1 coincides with γ.

We are now ready to give our main definition.

Definition 3.2. Given a fixed positive integer k, the *shrinking factor*, a positive integer x in the interval $\left[2^{hk}, 2^{(h+1)k} - 1\right]$ is ζ_k-coded by writing $h+1$ in unary, followed by a minimal binary code of $x - 2^{hk}$ in the interval $\left[0, 2^{(h+1)k} - 2^{hk} - 1\right]$.

Table 2 shows the first sixteen codewords of some ζ codes and of some other common codes.

The first 2^{hk} integers in the range $\left[2^{hk}, 2^{(h+1)k} - 1\right]$ are coded using $(h+1)\times(k+1)-1$ bits, and the remaining ones are coded using $(h+1)(k+1)$ bits. This observation yields easily the implied distribution of ζ_k:

$$2^{-\lfloor(\log x)/k+1\rfloor(k+1)+\tau(x)} \approx \frac{1+\tau(x)}{x^{1+\frac{1}{k}}}$$

where

$$\tau(x) = \begin{cases} 0 & \text{if } (\log x)/k - \lfloor(\log x)/k\rfloor \in [0, 1/k) \\ 1 & \text{otherwise.} \end{cases}$$

The above distribution is very close to a power law with exponent $1 + \frac{1}{k}$. A simple computation shows that ζ codes are complete, which makes them optimal for the above distribution and *a fortiori* a good candidate for integers distributed following a power-law with exponent smaller than two. Nonetheless, as we already remarked, the approximations introduced by removing the floor/ceiling operators may produce misleading results. Thus, in the rest of the paper, we attack the problem of giving precise estimates of the performance of γ, δ, ζ, and variable-length block codes with respect to power laws with small exponent.

4. Preliminaries

The main difficulty in estimating the expected length of a code with respect to a power-law distribution is that the random variable giving the code length has a countable number of discontinuities: for instance, the expected length of γ code is given by

$$\frac{1}{\zeta(\alpha)}\sum_{x=1}^{\infty}(2\lfloor\log x\rfloor+1)\frac{1}{x^{\alpha}},$$

and the floor gets in the way.[7] Fortunately, there is a better way to express the same summation—we can sum the length of all codes with a given unary prefix, and then sum over all unary prefixes:

$$\sum_{h=0}^{\infty}(2h+1)\sum_{x=2^h}^{2^{h+1}-1}\frac{1}{\zeta(\alpha)x^{\alpha}}.$$

Of course, we have now the problem of estimating the inner sums, but we shall use for that purpose Euler's powerful summation formula, which in our case is particularly well-behaved.

Recall Euler's summation formula [Graham et al. 94]:

$$\sum_{x=a}^{b-1}f(x)=\int_a^b f(x)\,dx-\frac{1}{2}f(x)\bigg|_a^b+\sum_{t=1}^{m}\frac{B_{2t}}{(2t)!}f^{(2t-1)}(x)\bigg|_a^b+R_{2m}$$

where B_i represents the ith Bernoulli number, $f^{(i)}$ denotes the ith derivative of f, and R_i is a remainder term. This formula holds for functions with enough derivatives, but the behaviour of the remainder term can vary wildly.

We shall apply this formula to the function $f(x)=x^{-\alpha}$. Observe that in this case $f^{(2t)}(x)>0$ and $f^{(2t+1)}(x)<0$, for all $x>0$ and $t\geq 0$. This implies [Graham et al. 94, page 461] that the remainder has the form

$$R_{2m}=\theta_{2m}\frac{B_{2m+2}}{(2m+2)!}\,f^{(2m+1)}(x)\bigg|_a^b,$$

[7]It would be tempting to just throw the floor away, but, as in the case of implied distributions, it is not a good idea; using the formulae we present, the reader will be able to check that this approximation would lead to a very large error.

with $0 < \theta_{2m} < 1$. Now, since B_{2m+2} is positive iff m is even, we have that R_{2m} is positive iff m is odd: in other words, remainders have alternating signs. We conclude that, for every odd $m > 0$,

$$\int_a^b f(x)\,dx - \frac{1}{2}f(x)\Big|_a^b + \sum_{t=1}^{m+1} \frac{B_{2t}}{(2t)!} f^{(2t-1)}(x)\Big|_a^b$$
$$\leq \sum_{x=a}^{b-1} f(x) \leq \int_a^b f(x)\,dx - \frac{1}{2}f(x)\Big|_a^b + \sum_{t=1}^{m} \frac{B_{2t}}{(2t)!} f^{(2t-1)}(x)\Big|_a^b .$$

In particular, we can bound the approximation error easily.

Let us consider now a recurring pattern in the following computations: we have a parameter h, and we have to approximate a summation of $x^{-\alpha}$ between c^h and $rc^h - 1$, where $r \geq 1$ is independent from h. In this case the (upper or lower, depending on the summation index) bound used in the last inequality becomes

$$\int_{c^h}^{rc^h} x^{-\alpha}\,dx - \frac{1}{2}x^{-\alpha}\Big|_{c^h}^{rc^h}$$
$$+\sum_{t=1}^{m} \frac{B_{2t}}{(2t)!}(-\alpha)(-\alpha-1)\dots(-\alpha-2t+2)x^{-\alpha-2t+1}\Big|_{c^h}^{rc^h} = \frac{1-r^{1-\alpha}}{\alpha-1}c^{h(1-\alpha)}$$
$$+\frac{1-r^{-\alpha}}{2}c^{-h\alpha} + \sum_{t=1}^{m}\frac{B_{2t}}{2t}\binom{\alpha+2t-2}{2t-1}\left(1-r^{-\alpha-2t+1}\right)c^{h(-\alpha-2t+1)}.$$

The latter form is particularly useful in our computations, as it makes it possible to sum easily over all $h \geq 0$, possibly multiplying first by some sequence of coefficients β_h. More explicitly,

$$\begin{aligned}
&\sum_h \beta_h \sum_{x=c^h}^{rc^h-1} x^{-\alpha} \\
&\quad\approx \sum_h \beta_h \Bigg[\frac{1-r^{1-\alpha}}{\alpha-1}c^{h(1-\alpha)} + \frac{1-r^{-\alpha}}{2}c^{-h\alpha} \\
&\qquad\qquad + \sum_{t=1}^{m}\frac{B_{2t}}{2t}\binom{\alpha+2t-2}{2t-1}\left(1-r^{-\alpha-2t+1}\right)c^{h(-\alpha-2t+1)}\Bigg] \\
&\quad= \frac{1-r^{1-\alpha}}{\alpha-1}\sum_h \beta_h\left(c^{1-\alpha}\right)^h + \frac{1-r^{-\alpha}}{2}\sum_h \beta_h\left(c^{-\alpha}\right)^h \\
&\qquad + \sum_{t=1}^{m}\frac{B_{2t}}{2t}\binom{\alpha+2t-2}{2t-1}\left(1-r^{-\alpha-2t+1}\right)\sum_h \beta_h\left(c^{-\alpha-2t+1}\right)^h .
\end{aligned}$$

Thus, when $c > 1$, usually playing a bit with geometric series is enough to get a closed form for the above summations.

Of course, the main decision to be taken is the value of m, which determines the gap between the lower and the upper bound. In general, we will use $m = 1$; in other words, we compute an upper bound by taking as the last summand in the estimate the single term

$$\frac{\alpha}{12}\left(1 - r^{-\alpha-1}\right)\sum_h \beta_h \left(c^{-\alpha-1}\right)^h .$$

In turn, this means that the gap between the upper and the lower bound (which can be used as an error estimate) is given by

$$\frac{\alpha(\alpha+1)(\alpha+2)}{720}\left(1 - r^{-\alpha-3}\right)\sum_h \beta_h \left(c^{-\alpha-3}\right)^h .$$

We shall estimate this error both absolutely and relative to the lower bound (which gives an upper bound on the real relative error). Note that by increasing m one can easily obtain more precise estimates, at the price of more complex formulae.

5. Estimating γ

We start by evaluating the expected length of the simplest code, Elias' γ. (We will obtain again this result for ζ_1, but it is useful as a warm-up.) Observing that integers in the range from 2^h to $2^{h+1}-1$ are represented by $2h+1$ bits, we have an expected length of

$$\sum_{h=0}^{\infty}(2h+1)\sum_{x=2^h}^{2^{h+1}-1}\frac{1}{\zeta(\alpha)x^\alpha}.$$

Our framework applies with $c = r = 2$ and $\beta_h = 2h+1$; since $\sum_{h=0}^{\infty}(2h+1)q^h = (1+q)/(1-q)^2$, we obtain

$$\frac{1}{\zeta(\alpha)}\left[\frac{1+2^{1-\alpha}}{(\alpha-1)(1-2^{1-\alpha})} + \frac{1+2^{-\alpha}}{2(1-2^{-\alpha})} + \frac{\alpha\left(1+2^{-\alpha-1}\right)}{12\left(1-2^{-\alpha-1}\right)}\right],$$

with an absolute error of

$$\frac{1}{\zeta(\alpha)}\left[\frac{\alpha(\alpha+1)(\alpha+2)(1+2^{-\alpha-3})}{720(1-2^{-\alpha-3})}\right] \leq 0.022 \text{ bits if } \alpha \leq 2,$$

which corresponds to a relative error of 9%.

6. Estimating Variable-Length Nibble Code

The variable-length nibble code is slightly irregular because of its redundancies, but we shall be able to reduce it to a standard treatment. Observing that 1 is coded with four bits, and that integers in the range from $2^{3h}+1$ to $2^{3(h+1)}$ are represented by $4(h+1)$ bits, we have an expected length of

$$\frac{4}{\zeta(\alpha)} + \sum_{h=0}^{\infty} 4(h+1) \sum_{x=2^{3h}+1}^{2^{3(h+1)}} \frac{1}{\zeta(\alpha)x^{\alpha}} =$$

$$\frac{4}{\zeta(\alpha)} + \sum_{h=0}^{\infty} 4(h+1) \sum_{x=2^{3h}}^{2^{3(h+1)}-1} \frac{1}{\zeta(\alpha)x^{\alpha}} - \sum_{h=0}^{\infty} \frac{4}{8^{\alpha h}\zeta(\alpha)}.$$

Our framework applies to the second summation with $c = r = 8$ and $\beta_h = 4(h+1)$; since $\sum_{h=0}^{\infty} 4(h+1)q^h = 4/(1-q)^2$ and the last summation is trivial, we obtain

$$\frac{4}{\zeta(\alpha)}\left[1 + \frac{1}{(\alpha-1)(1-8^{1-\alpha})} + \frac{1}{2(1-8^{-\alpha})} + \frac{\alpha}{12\,(1-8^{-\alpha-1})} + \frac{1}{1-8^{-\alpha}}\right],$$

with an absolute error of

$$\frac{1}{\zeta(\alpha)}\left[\frac{\alpha(\alpha+1)(\alpha+2)}{180(1-8^{-\alpha-3})}\right] \leq 0.081 \text{ bits if } \alpha \leq 2,$$

which corresponds to a relative error of 9%.

7. Estimating ζ_k

Observing that integers in the range $\left[2^{hk}, 2^{hk+1}-1\right]$ are coded using $(h+1)(k+1)-1$ bits, and integers in the range $\left[2^{hk+1}, 2^{(h+1)k}-1\right]$ are coded using $(h+1)(k+1)$ bits, the expected code length is

$$\sum_{h=0}^{\infty}\left\{\sum_{x=2^{hk}}^{2^{hk+1}-1}[(h+1)(k+1)-1]\frac{1}{\zeta(\alpha)x^{\alpha}} + \sum_{x=2^{hk+1}}^{2^{(h+1)k}-1}[(h+1)(k+1)]\frac{1}{\zeta(\alpha)x^{\alpha}}\right\}$$

$$= (k+1)\sum_{h=0}^{\infty}\sum_{x=2^{hk}}^{2^{(h+1)k}-1}(h+1)\frac{1}{\zeta(\alpha)x^{\alpha}} - \sum_{h=0}^{\infty}\sum_{x=2^{hk}}^{2^{hk+1}-1}\frac{1}{\zeta(\alpha)x^{\alpha}}.$$

The framework described above applies to the *first summation* with $c = r = 2^k$ and $\beta_h = h+1$; since $\sum_{h=0}^{\infty}(h+1)q^h = 1/(1-q)^2$, we obtain

$$\frac{k+1}{\zeta(\alpha)}\left[\frac{1}{(\alpha-1)(1-2^{k(1-\alpha)})} + \frac{1}{2(1-2^{-k\alpha})} + \frac{\alpha}{12(1-2^{k(-\alpha-1)})}\right].$$

The absolute error is bounded by

$$\frac{k+1}{\zeta(\alpha)}\left[\frac{\alpha(\alpha+1)(\alpha+2)}{720(1-2^{k(-\alpha-3)})}\right] \leq 0.142 \text{ bits if } \alpha \leq 2 \text{ and } k < 7.$$

For the *second summation* we apply again our framework with $c = 2^k$, $r = 2$, and $\beta_h = 1$, getting

$$\frac{1}{\zeta(\alpha)}\left[\frac{1-2^{1-\alpha}}{(\alpha-1)(1-2^{k(1-\alpha)})} + \frac{1-2^{-\alpha}}{2(1-2^{-k\alpha})} + \frac{\alpha(1-2^{-\alpha-1})}{12(1-2^{k(-\alpha-1)})}\right],$$

with an absolute error of

$$\frac{1}{\zeta(\alpha)}\left[\frac{\alpha(\alpha+1)(\alpha+2)(1-2^{-\alpha-3})}{720(1-2^{k(-\alpha-3)})}\right] \leq 0.020 \text{ bits if } \alpha \leq 2 \text{ and } k < 7.$$

The combined absolute errors produce (in the stated range) a relative error of 21%. We note, however, that in the range we will be analysing, that is, $\alpha \leq 1.6$, the relative error reduces to 9%.

8. Estimating δ

We conclude our tour by approximating the expected length of δ code. This case turns out to be the most difficult one because of the three-level structure of the code, which is reflected by the triple summation expressing its expected length:

$$\sum_{h=0}^{\infty}\sum_{k=2^h}^{2^{h+1}-1}(2h+k)\sum_{x=2^{k-1}}^{2^k-1}\frac{1}{\zeta(\alpha)x^\alpha}.$$

To attack the problem, we first divide the estimate into a part that fits our framework and a part that needs separate treatment:

$$\begin{aligned}
&\sum_{h=0}^{\infty}\sum_{k=2^h}^{2^{h+1}-1}(2h+k)\sum_{x=2^{k-1}}^{2^k-1}\frac{1}{\zeta(\alpha)x^\alpha}\\
&\qquad=\sum_{h=0}^{\infty}\sum_{k=2^h}^{2^{h+1}-1}2h\sum_{x=2^{k-1}}^{2^k-1}\frac{1}{\zeta(\alpha)x^\alpha}+\sum_{h=0}^{\infty}\sum_{k=2^h}^{2^{h+1}-1}k\sum_{x=2^{k-1}}^{2^k-1}\frac{1}{\zeta(\alpha)x^\alpha}\\
&\qquad=\sum_{h=0}^{\infty}2h\sum_{x=2^{2^h-1}}^{2^{2^{h+1}-1}-1}\frac{1}{\zeta(\alpha)x^\alpha}+\sum_{h=0}^{\infty}(h+1)\sum_{x=2^h}^{2^{h+1}-1}\frac{1}{\zeta(\alpha)x^\alpha}.
\end{aligned}$$

The last sum is by now familiar, and by setting $c = r = 2$, $\beta_h = h + 1$, our framework yields

$$\frac{1}{\zeta(\alpha)}\left[\frac{1}{(\alpha-1)(1-2^{1-\alpha})}+\frac{1}{2(1-2^{-\alpha})}+\frac{1}{12\,(1-2^{-\alpha-1})}\right], \tag{8.1}$$

with an absolute error of

$$\frac{1}{\zeta(\alpha)}\left[\frac{\alpha(\alpha+1)(\alpha+2)}{720(1-2^{-\alpha-3})}\right] \le 0.021 \text{ bits if } \alpha \le 2.$$

The first sum, instead, requires a more careful treatment. By the same argument as in Section 4, we are interested in evaluating

$$\begin{aligned}
&\int_{2^{2^h-1}}^{2^{2^{h+1}-1}} x^{-\alpha}\,dx - \frac{1}{2}x^{-\alpha}\Big|_{2^{2^h-1}}^{2^{2^{h+1}-1}} - \frac{\alpha}{12}x^{-\alpha-1}\Big|_{2^{2^h-1}}^{2^{2^{h+1}-1}}\\
&= \frac{1}{\alpha-1}\left[\left(2^{1-\alpha}\right)^{2^h-1}-\left(2^{1-\alpha}\right)^{2^{h+1}-1}\right]+\frac{1}{2}\left[\left(2^{-\alpha}\right)^{2^h-1}-\left(2^{-\alpha}\right)^{2^{h+1}-1}\right]\\
&\quad+\frac{\alpha}{12}\left[\left(2^{-\alpha-1}\right)^{2^h-1}-\left(2^{-\alpha-1}\right)^{2^{h+1}-1}\right]\\
&= \frac{2^{\alpha-1}}{\alpha-1}\left[\left(2^{1-\alpha}\right)^{2^h}-\left(2^{1-\alpha}\right)^{2^{h+1}}\right]+2^{\alpha-1}\left[\left(2^{-\alpha}\right)^{2^h}-\left(2^{-\alpha}\right)^{2^{h+1}}\right]\\
&\quad+\frac{\alpha 2^{\alpha}}{6}\left[\left(2^{-\alpha-1}\right)^{2^h}-\left(2^{-\alpha-1}\right)^{2^{h+1}}\right].
\end{aligned}$$

The next summand, which we will use to derive the error bound, is

$$-\frac{\alpha(\alpha+1)(\alpha+2)2^{\alpha}}{90}\left[\left(2^{-\alpha-3}\right)^{2^h}-\left(2^{-\alpha-3}\right)^{2^{h+1}}\right].$$

Now, letting

$$F(x)=\sum_{h\ge 0} h\,x^{2^h}-\sum_{h\ge 0} h\,x^{2^{h+1}},$$

we can give, as we did previously, the estimate for the second summation:

$$\frac{1}{\zeta(\alpha)}\left[\frac{2^{\alpha}}{\alpha-1}F\left(2^{1-\alpha}\right)+2^{\alpha}F\left(2^{-\alpha}\right)+\frac{\alpha 2^{\alpha+1}}{6}F\left(2^{-\alpha-1}\right)\right], \tag{8.2}$$

with its absolute error bound

$$\frac{1}{\zeta(\alpha)}\left[\frac{\alpha(\alpha+1)(\alpha+2)2^{\alpha}}{45}F\left(2^{-\alpha-3}\right)\right].$$

Getting sound numerical data from $F(x)$ is going to be tough. Let us define

$$f(x)=\sum_{h\ge 0} x^{2^h}$$

and note that

$$\begin{aligned} F(x) &= \sum_{h\geq 0} h\, x^{2^h} - \sum_{h\geq 0} h\, x^{2^{h+1}} \\ &= (x^2 + 2x^4 + 3x^8 + \dots) - (x^4 + 2x^8 + 3x^{16} + \dots) \\ &= f(x) - x. \end{aligned}$$

The series $\sum_{h\geq 0} x^{2^h}$ is a well-known pathological object, usually found in analysis textbooks [Rudin 86]. Albeit it can be shown that $f(x)$ cannot have a finite algebraic expression in the standard transcendental functions (log, sin, cos, etc.), numerically the following inequalities hold in $[0, 1)$ with a negligible error (γ denotes Euler's constant):

$$-\log(1-x) + \left(\frac{1}{2} - \frac{\gamma}{\ln 2}\right)\log(1+x) \leq f(x) \leq -\log(1-x) + \left(\frac{1}{2} - \frac{\gamma}{\ln 2}\right) x.$$

These inequalities, when $x \to 1^-$, still make sense; more precisely, Giuseppe Molteni [Molteni 04] showed that

$$\limsup_{x\to 1^-} \left| f(x) + \log(1-x) - \frac{1}{2} + \frac{\gamma}{\ln 2} \right| < 4.5 \cdot 10^{-6}.$$

All in all, we can upper bound the expected length by plugging into Equation (8.2) the upper bound for $F(x)$ and adding Equation (8.1); of course, when bounding the relative error we need *lower* bounds for the denominator, hence the need for both inequalities for $f(x)$. The resulting relative error is bounded by 68% when $\alpha \leq 2$, albeit for $\alpha < 1.6$ it reduces to 27%.

9. Comparing Codes

The usefulness of the formulae described in Table 3 lies in the possibility of comparing the codes with respect to the value of α, hence choosing the best code for a specific α.

It is easy to check that variable-length nibble code is in general not a good candidate because of its redundancy (unless four-bit alignment is a requirement), as shown in Figure 5. However, its behaviour is close to that of ζ_3, which explains its success in the LINK database.

Figure 3 is in our opinion the best graphical representation of the formulae of Table 3. It displays three surfaces, representing the expected lengths of γ, δ, and ζ_k codes with respect to a power-law distribution with exponent α. For ζ_k, the expected length depends of course on k, whereas the other two surfaces have no dependency on k.

Code	Expected length (upper bound)
γ	$\frac{1}{\zeta(\alpha)}\left[\frac{1+2^{1-\alpha}}{(\alpha-1)(1-2^{1-\alpha})}+\frac{1+2^{-\alpha}}{2(1-2^{-\alpha})}+\frac{\alpha\left(1+2^{-\alpha-1}\right)}{12\,(1-2^{-\alpha-1})}\right]$
nibble	$\frac{4}{\zeta(\alpha)}\left[1+\frac{1}{(\alpha-1)(1-8^{1-\alpha})}+\frac{3}{2(1-8^{-\alpha})}+\frac{\alpha}{12\,(1-8^{-\alpha-1})}\right]$
ζ_k	$\frac{1}{\zeta(\alpha)}\left[\frac{k+2^{1-\alpha}}{(\alpha-1)(1-2^{k(1-\alpha)})}+\frac{k+2^{-\alpha}}{2(1-2^{-k\alpha})}+\frac{\alpha(k+2^{-\alpha-1})}{12(1-2^{k(-\alpha-1)})}\right]$
δ	$\frac{1}{\zeta(\alpha)}\Big\{\frac{1}{(\alpha-1)(1-2^{1-\alpha})}+\frac{1}{2(1-2^{-\alpha})}+\frac{1}{12\,(1-2^{-\alpha-1})}+ +\frac{2^\alpha}{\alpha-1}\left[-\log(1-2^{1-\alpha})-\left(\frac{1}{2}+\frac{\gamma}{\ln 2}\right)2^{1-\alpha}\right]+ +2^\alpha\left[-\log(1-2^{-\alpha})-\left(\frac{1}{2}+\frac{\gamma}{\ln 2}\right)2^{-\alpha}\right]+ +\frac{\alpha 2^{\alpha+1}}{6}\left[-\log(1-2^{-\alpha-1})-\left(\frac{1}{2}+\frac{\gamma}{\ln 2}\right)2^{-\alpha-1}\right]\Big\}$

Table 3. A table of approximated lengths.

α	Code
< 1.06	δ (see remarks)
$[1.06, 1.08]$	ζ_6
$[1.08, 1.11]$	ζ_5
$[1.11, 1.16]$	ζ_4
$[1.16, 1.27]$	ζ_3
$[1.27, 1.57]$	ζ_2
$[1.57, 2]$	$\gamma=\zeta_1$

Table 4. Suggested ranges for codes.

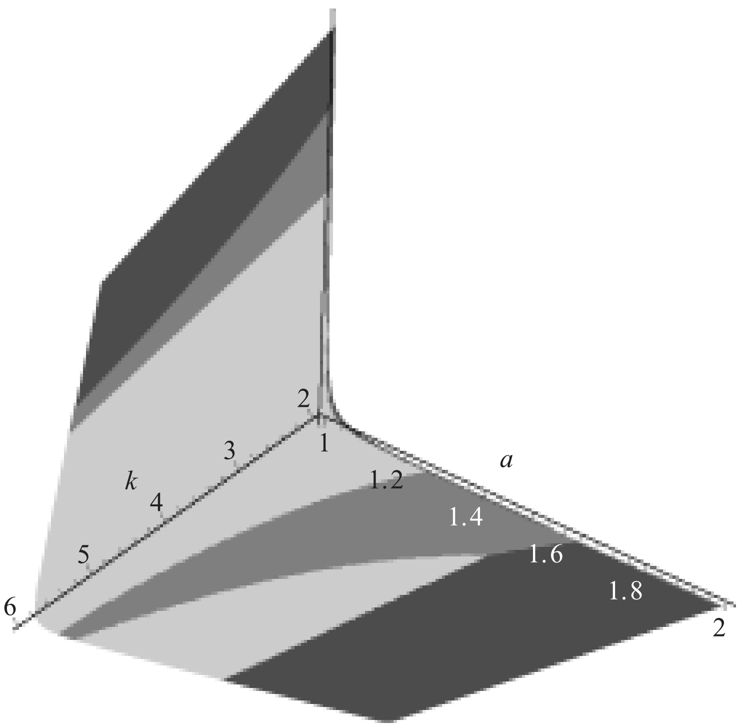

Figure 3. A three-dimensional graph representing the expected length of γ (dark gray), δ (light gray), and ζ (medium gray) codes (only the surface corresponding to ζ codes depends on k). Note that we are looking at the graph from below, so the horn-shaped visible part of the ζ_k surface highlights the region where ζ codes are more efficient than γ or δ codes.

The figure clearly shows that for values of α larger than 1.57, γ code is the best choice. On the other hand, for all smaller values of α, there is always a k for which ζ_k has a better behaviour than δ (look at the horn-shaped part of the ζ_k surface that protrudes below the δ and γ surfaces). However, when α is, say, smaller than 1.05, the advantage is unlikely to show in real applications because of the limited number of codewords actually used. This situation is summarized in Table 4, which suggests the right ranges for the first ζ codes, and in Figure 4, where we compare the redundancy of δ and ζ codes.

Our estimates also prove that the implied distributions given in Section 3 for ζ codes and variable-length block codes are quite imprecise: for instance, one would falsely believe that the best ζ code for $\alpha \approx 1.2 = 1 + \frac{1}{4}$ is ζ_4, whereas ζ_3 is the right one.

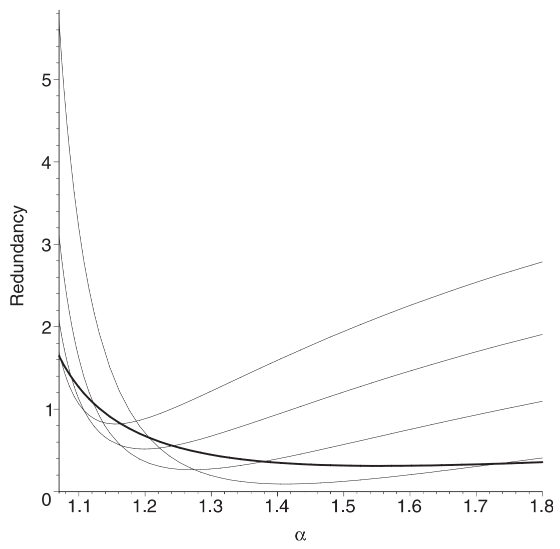

Figure 4. A graph representing the redundancy, that is, the difference between expected length and entropy, of δ (thick line) and ζ codes (ζ_5 to ζ_2, in increasing order of their minima) when $1 \leq \alpha \leq 1.8$. Note that δ crosses all other lines.

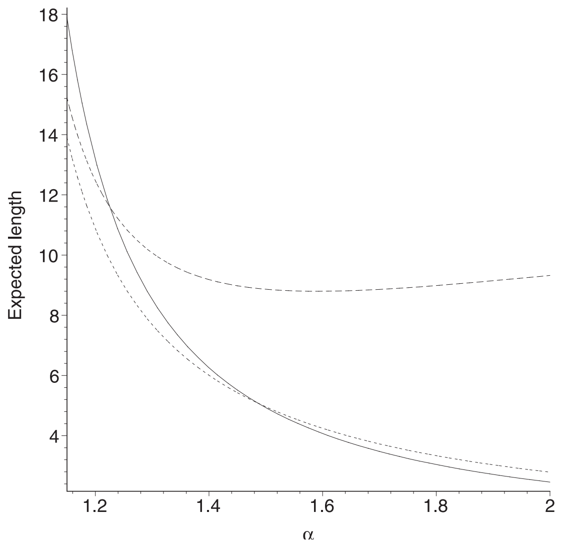

Figure 5. A graph comparing the expected lengths of γ (continuous line), δ (dashed line), and variable-length nibble codes (dotted line) when $1.15 \leq \alpha \leq 2$. Variable-length nibble code is always worse than δ (outside of this small range there are no other intersections).

	Huffman code			ζ codes			
α	Entropy	Exp. length	Red.	H_α	Code	Exp. length	Red.
1.15	11.37	11.39	0.23%	13.01	ζ_4	13.61	4.66%
1.20	9.62	9.66	0.46%	10.20	ζ_3	10.57	3.56%
1.25	8.24	8.27	0.37%	8.45	ζ_3	8.72	3.21%
1.30	7.16	7.19	0.28%	7.24	ζ_2	7.44	2.69%
1.35	6.32	6.34	0.36%	6.35	ζ_2	6.47	1.86%
1.40	5.64	5.68	0.63%	5.65	ζ_2	5.75	1.65%
1.45	5.10	5.16	1.18%	5.10	ζ_2	5.20	1.95%
1.50	4.64	4.70	1.16%	4.64	ζ_2	4.77	2.68%
1.55	4.26	4.29	0.82%	4.26	ζ_2	4.42	3.79%
1.60	3.93	3.95	0.60%	3.93	ζ_1	4.07	3.61%

Table 5. Comparison between Huffman codes (for 10^9 codewords) and ζ codes. The entropy column for Huffman codes refers to a truncated power-law. The expected length for ζ codes is computed using the estimates given in Table 3. In both cases, the last column shows the relative redundancy.

As we already noticed, Huffman codes are impractical for the compression of web graphs, due to the large number of codewords.[8] Nonetheless, a comparison with Huffman codes is instructive. Table 5 presents a comparative analysis of ζ codes with respect to minimum-redundancy Huffman codes (computed using Moffat and Katajainen's in-place minimum-redundancy coder [Moffat and Katajainen 95]). For each value of α, we have generated 10^9 codewords, distributed as a (truncated) power law with exponent α. The first three columns show the entropy, the average length of Huffman codes, and the corresponding redundancy (relative to the entropy). Then, the table shows the theoretical entropy H_α, the ζ code suitable for that value of α (according to Table 4), its expected length, and its redundancy (relative to H_α). Note that the theoretical entropy is larger than the actual entropy and that the difference is significant for small values of α, because in that case the distribution decreases more slowly and the codewords after 10^9 give a nonnegligible contribution.

However, this comparison is of purely theoretical interest, since storing the coding table for 10^9 codewords is not feasible (and would exceed by an order of magnitude the graph size—see [Boldi and Vigna 04]).

[8]Of course, one could use a hybrid scheme that Huffman-encodes an initial segment of the integers and then performs some sort of escaping toward a simpler code; some experiments in this sense has been performed in [Randall et al. 01], but the trade-off between speed and size was judged unsatisfactory.

18.5 Mpages, 300 Mlinks from .uk			118 Mpages, 1 Glinks from WebBase		
Code	# bits/node	# bits/link	Code	# bits/node	# bits/link
Standard compression			Standard compression		
γ	40.19	2.50	γ	29.93	3.47
δ	37.15	2.31	δ	27.09	3.14
nibble	36.44	2.26	nibble	26.99	3.13
ζ_2	36.24	2.25	ζ_2	26.93	3.12
ζ_3	**35.81**	**2.22**	**ζ_3**	**26.57**	**3.08**
ζ_4	36.03	2.24	ζ_4	26.83	3.11
ζ_5	36.70	2.28	ζ_5	27.41	3.17
No differential compression			No differential compression		
γ	139.81	8.69	γ	73.60	8.53
δ	125.39	7.79	δ	65.52	7.59
nibble	121.19	7.53	nibble	63.83	7.39
ζ_2	120.58	7.49	ζ_2	63.62	7.37
ζ_3	**118.47**	**7.36**	**ζ_3**	**62.43**	**7.23**
ζ_4	120.30	7.47	ζ_4	63.43	7.35
ζ_5	124.20	7.72	ζ_5	65.58	7.60
Gap compression only			Gap compression only		
γ	134.67	8.37	γ	71.40	8.27
δ	120.50	7.49	**δ**	**63.46**	**7.35**
nibble	129.99	8.08	nibble	70.06	8.11
ζ_2	**119.76**	**7.44**	ζ_2	64.05	7.42
ζ_3	122.54	7.61	ζ_3	65.82	7.62
ζ_4	129.62	8.05	ζ_4	69.97	8.11
ζ_5	138.90	8.63	ζ_5	75.29	8.72
Transpose, standard compression			Transpose, standard compression		
γ	35.31	2.19	γ	27.76	3.22
δ	33.22	2.06	δ	25.60	2.97
nibble	32.53	2.02	nibble	25.37	2.94
ζ_2	31.97	1.99	ζ_2	25.16	2.91
ζ_3	**31.88**	**1.98**	**ζ_3**	**24.96**	**2.89**
ζ_4	32.32	2.01	ζ_4	25.32	2.93
ζ_5	33.20	2.06	ζ_5	25.98	3.01
Transpose, no differential compression			Transpose, no differential compression		
γ	49.27	3.06	γ	39.31	4.55
δ	46.13	2.87	δ	36.01	4.17
nibble	44.62	2.77	nibble	34.99	4.05
ζ_2	43.80	2.72	ζ_2	34.69	4.02
ζ_3	**43.64**	**2.71**	**ζ_3**	**34.29**	**3.97**
ζ_4	44.52	2.77	ζ_4	34.90	4.04
ζ_5	46.10	2.86	ζ_5	36.09	4.18
Transpose, gap compression only			Transpose, gap compression only		
γ	55.83	3.47	γ	41.00	4.75
δ	**52.87**	**3.28**	**δ**	**37.82**	**4.38**
nibble	86.78	5.39	nibble	53.31	6.18
ζ_2	62.00	3.86	ζ_2	41.76	4.84
ζ_3	73.92	4.59	ζ_3	47.06	5.45
ζ_4	87.03	5.41	ζ_4	53.47	6.19
ζ_5	100.87	6.27	ζ_5	60.47	7.00

Table 6. Comparative compression on an actual web graph. The best code is shown in boldface.

10. Experimental Results

We conclude by presenting a table of experimental data.[9] Table 6 reports the number of bits per link and per node obtained by compressing two sample graphs and their transposes using WebGraph.

To make the data completely understandable, we must recall some of the compression techniques used by WebGraph [Boldi and Vigna 04]. *Differential compression* represents the successor list of x by copying part of the list from some node $y < x$. The copy is represented by a sequence of suitably-coded integers representing inclusion-exclusion blocks (copy this number of successors, skip this number of successors, copy again, etc.).

The rest of the successor list (i.e., the successors of x that are not successors of y) might also be *intervalised*. Namely, if intervals of consecutive integers longer than a given threshold are present, they are converted into a sequence of pairs formed by the left extreme and by the length of the interval. Finally, the remaining successors are stored by recording their gaps, using the code shown in Table 6. For more details, see [Boldi and Vigna 04].

The first group of data in Table 6 is gathered using standard compression, that is, default values for all parameters, which amounts to using both differential compression and intervalisation. The second group is obtained by turning off differential compression but keeping intervalisation. The third group uses just gap compression, as it happens in the statistics shown in Figures 1 and 2.

The results are quite interesting. The best codes are almost always ζ_2 and ζ_3, in agreement with the theory. In three cases, if no intervalisation is applied, δ turns out to be the better code. This should not surprise the reader: as we noted in Section 2, the gap of one lies out of the distribution, and the fact that it is coded by a single bit in δ code explains why the latter performs better. However, the resulting number of bits per link is much worse than the one obtained using intervalisation, which records separately subsequences of consecutive integers: by eliminating the gap of one, the gap distribution gets smoothed, and it fits a power-law distribution more precisely.

References

[Adler and Mitzenmacher 01] Micah Adler and Michael Mitzenmacher. "Towards Compressing Web Graphs." In *2001 Data Compression Conference: March 27–*

[9]The .uk data were gathered using UbiCrawler; the WebBase data refer to the 1/2001 general crawl.

29, 2001 in Snowbird, Utah: Proceedings, edited by James A. Storer and Martin Cohn, pp. 203–212. Los Alamitos, CA: IEEE Press, 2001.

[Boldi and Vigna 04] Paolo Boldi and Sebastiano Vigna. "The WebGraph Framework I: Compression Techniques." In *Proceedings of the Thirteenth International Conference on World Wide Web*, pp. 595–601. New York: ACM Press, 2004.

[Boldi et al. 04] Paolo Boldi, Bruno Codenotti, Massimo Santini, and Sebastiano Vigna. "UbiCrawler: A Scalable Fully Distributed Web Crawler." *Software: Practice & Experience* 34:8 (2004), 711–726.

[Broder et al. 00] Andrei Broder, Ravi Kumar, Farzin Maghoul, Prabhakar Raghavan, Sridhar Rajagopalan, Raymie Stata, Andrew Tomkins, and Janet Wiener. "Graph Structure in the Web: Experiments and Models." *Computer Networks* 33:1–6 (2000), 309–320.

[Elias 75] Peter Elias. "Universal Codeword Sets and Representations of the Integers." *IEEE Transactions on Information Theory* 21 (1975), 194–203.

[Graham et al. 94] Ronald L. Graham, Donald E. Knuth, and Oren Patashnik. *Concrete Mathematics*, Second edition. Reading, MA: Addison–Wesley, 1994.

[Hirai et al. 00] Jun Hirai, Sriram Raghavan, Hector Garcia-Molina, and Andreas Paepcke. "WebBase: A Repository of Web Pages." *Computer Networks* 33:1–6 (2000), 277–293.

[Levenstein 68] V. E. Levenstein. "On the Redundancy and Delay of Separable Codes for the Natural Numbers." *Problems of Cybernetics* 20 (1968), 173–179.

[Moffat and Katajainen 95] Alistair Moffat and Jyrki Katajainen. "In-Place Calculation of Minimum-Redundancy Codes." In *Alogrithms and Data Structures: 4th International Workshop, WADS '95, Kingston, Canada August 16–18, 1995: Proceedings*, edited by S. G. Akl, F. Dehne, and J.-R. Sack, pp. 393–402. Lecture Notes in Computer Science 955. New York: Springer, 1995.

[Molteni 04] Giuseppe Molteni. "On a Class of Lacunary Series." Preprint, 2004.

[Randall et al. 01] Keith Randall, Raymie Stata, Rajiv Wickremesinghe, and Janet L. Wiener. "The LINK Database: Fast Access to Graphs of the Web." Research Report 175, Compaq Systems Research Center, Palo Alto, CA, 2001.

[Rudin 86] Walter Rudin. *Real and Complex Analysis*, Third edition. New York: McGraw–Hill, 1986.

[Witten et al. 99] Ian H. Witten, Alistair Moffat, and Timothy C. Bell. *Managing Gigabytes: Compressing and Indexing Documents and Images*, Second edition. Los Altos, CA: Morgan Kaufmann Publishers, 1999.

Paolo Boldi, Dipartimento di Scienze dell'Informazione, Università degli Studi di Milano, via Comelico 39/41, I-20135 Milano, Italy (boldi@dsi.unimi.it)

Sebastiano Vigna, Dipartimento di Scienze dell'Informazione, Università degli Studi di Milano, via Comelico 39/41, I-20135 Milano, Italy. (vigna@acm.org)

Received May 17, 2004; accepted December 4, 2004.