Towards Analytical Information Theory: Recent Results on Lempel-Ziv Data Compression Schemes

Wojciech Szpankowski
Purdue University

September 23, 1996

[summary by Philippe Flajolet]

Abstract

The Lempel-Ziv algorithms are well-known dynamic dictionary algorithms of use in data compression. The talk shows how analytic models originally developed for the analysis of tries and digital search trees may be used to characterize their compression characteristics.

1. The Lempel-Ziv algorithms

A *text* to be compressed is always a message to be transmitted (to your friend, to your laser printer) that has a "meaning", hence a certain structure that goes along with some sort of "redundancy". In a natural language like English, trigrams like 'ted' or 'ing' are much more likely to be encountered, and much more frequently so, than 'qrm' or 'bzw', and a text on data compression is likely to contain more repetitions of 'algorithms' and fewer of 'smoking' than a text on public health. Compression algorithms precisely try to capture such regularities.

These observations have given rise to a first generation of methods. For a text in English, list all the conceivable trigrams (say!) in the language and transmit trigrams codes instead of individual letters. In addition, shorter codes may be assigned to more frequent trigrams, yielding further gains—this can be done efficiently by Huffman's algorithm. These methods are known as "static dictionary algorithms". They have the obvious drawback of not being adaptive; a scheme originally designed for English text is not likely to accommodate well Sanskrit epics, postscript code, or image bitmaps.

Such was essentially the state of the art before the appearance of the celebrated Lempel-Ziv (LZ) papers in 1977 and 1978; see [10, 11]. The LZ algorithms—there are two basic ones and a denumerable collection of variants—can be viewed as building adaptively a "dynamic dictionary" that is dependent upon the particular text subjected to compression. Their common basis is:

THE "DEJA VU" PRINCIPLE.

(Pronounce as "day-jah voo"!) As the text proceeds, it is parsed into *segments* also called *phrases*. Instead of transmitting the letters of the text itself, just transmit *references* to places in the text where each segment was encountered before.

More precisely, the LZ77 and LZ78 algorithms, as considered here, are defined by:

LZ77. Scan the text. Starting a new segment, search for the longest matching factor already encountered in the past. Transmit its reference (position and length) and the next letter.

LZ78. Scan the text. Starting a new segment, search for the longest matching segment already encountered in the past. Transmit its reference (rank) and the next letter.

For instance, given a text that is a long sequence of a's, the two parsings start like

LZ77: | a | aa | aaaa | aaaaaaaa | ···
LZ78: | a | aa | aaa | aaaa | aaaaa | ···

In other words, LZ77 defines the new segment as anything (of maximal length) that has already been seen before, possibly across boundaries of previously defined segments, while LZ78 respects the boundaries of previously defined segments.

For LZ77, the transmission of each segment is of the type `<position, length, letter>` (position of the previous occurrence and length of the factor, plus new letter). For LZ78, it is of the type `<rank, letter>` (rank of the already encountered segment, plus new letter). So, in the case of our long string of a's, the next segment formed is encoded as

LZ77: `<1, 15, a>`, LZ78: `<5, a>`.

For LZ77, this says: "repeat the 15 characters starting at position 1 in the text and append an a". For LZ78: "repeat the 5th segment already parsed and append an a". Reconstruction of the source text at the receiving end is then particularly easy as it suffices to "expand" the references.

Another example is provided by the two sequences

LZ77: `<0,0,a> <0,0,b> <0,0,r> <1,1,c> <1,1,d> <1,4,#>`
LZ78: `<0,a> <0,b> <0,r> <1,c> <1,d> <1,b> <3,a> <0,#>`

that encode 'abracadabra# (with a terminator symbol). As these examples demonstrate, the LZ77 algorithm "learns" faster but, the implied dictionary being larger, references are more costly as their encodings require more bits. Fine analysis is thus needed in order to characterize the various tradeoffs involved.

Variants and implementations. There exist a great many variants of the LZ77 and LZ78 algorithms.

LZ77: One can consider that the entire alphabet sequence is prepended to the text. In this case, the new segments need not include a new-letter field, as each letter at least has been already encountered before. Also, one can limit the past fraction of the text to which the "deja vu" principle is applied (historically, to 8192 characters). One obtains in this way an algorithm close to the original LZ77 specification.

LZ78: Similarly, one can update the dictionary by deleting old segments (say, on a least recently used basis), and thus maintain a dictionary of an *a priori* bounded size.

In **LZ77**, when a new segment is started at position $(n+1)$ in the text, one must look for a longest factor that has occurred before. This corresponds to a virtual dictionary $\mathcal{D}$ that should contain all the $\frac{1}{2}n(n+1)$ factors of the text, as seen so far. There are two immediate solutions to this problem: (i) don't store the dictionary and use naïve string searching in the text itself; (ii) build a digital tree (also known as *trie*) of all suffixes of the part of the text seen so far. The time/space complexity pairs of these solutions are $\langle O(n^2), O(n)\rangle$ and $\langle O(n^2), O(n^2)\rangle$, respectively. The trie solution is interesting since a construction (the *suffix* tree) is known in order to eliminate duplication of informations; the suffix tree implementation has an $\langle O(n), O(n)\rangle$ complexity but is somewhat intricate to implement.

In contrast, **LZ78** is much easier to implement. It suffices to maintain a digital search tree of all the segments encountered so far. A new segment is detected by following a path in the tree. Each

internal node of the tree is associated with such a segment seen in the past and insertion takes place at an external node, which corresponds to extending a previously encountered segment.

These algorithms are famous. They are used in the Unix `compress` and `zip` commands (based on LZ78 and LZ77, resp.), in data transmission (the V42bis standard for modems), in image compression (the `gif` format), etc. Jump to your favourite web search engine for details.

2. Models and analysis

Classical information theory teaches us that the best rate at which a text can be compressed is given by the *entropy* function. Here, we concentrate on a binary memoryless channel, where each character in the source text has probability p of being a 0 and probability $q = 1 - p$ of being a 1. In that case, the (binary) entropy is

$$h = p \log_2 \frac{1}{p} + q \log_2 \frac{1}{q}. \tag{1}$$

A random text of n characters can then be compressed into *no less* than

$$h \cdot n(1 + o(1)) \tag{2}$$

bits, on average. The *redundancy* of a compression scheme is a measure of its distance to the information-theoretic lower bound (2).

It is already known that the two Lempel-Ziv algorithms achieve asymptotically the lower bound, so that the redundancy per character is $o(1)$ for both schemes. In other words, what is needed for further comparison is a *second order* asymptotic analysis of these algorithms. The talk centers on the LZ78 algorithm that has a mathematically pleasant decomposable structure closely related to *digital search trees*. Digital search tree intervene both as a data structure in the implementation of LZ78 and as a probabilistic model of random trees.

Digital search trees. Recall that a *digital search tree* or DST is a hybrid of the digital trie and the binary search tree defined as follows. A *sequence* $S = (s_1, \ldots, s_m)$ of m binary strings is given. The digital search tree $\mathrm{dst}(s)$ is recursively defined by: (i) the root contains the first string s_1; (ii) the left subtree is formed by taking the subsequence $S^{[0]}$ of $(s_2, \ldots, s_m)$ of strings starting with a 0 and stripped of this initial 0; (iii) the right subtree is formed similarly from strings starting with a 1. In other words,

$$\mathrm{dst}(S) = \langle s_1, \mathrm{dst}(S^{[0]} \backslash 0), \mathrm{dst}(S^{[1]} \backslash 1) \rangle,$$

where $S \backslash j$ means the sequence S with all its elements stripped of their initial letter j, and with $\mathrm{dst}(s) = \langle s, \emptyset, \emptyset \rangle$ for a one element sequence.

Assume that strings obey the Bernoulli model where each component bit b satisfies

$$\Pr\{b = 0\} = p, \qquad \Pr\{b = 1\} = q = 1 - p,$$

independently of the others. This model implies the *random DST model* [9], where a tree of size m has a left subtree of size K and a right subtree of size $m - 1 - K$ satisfying

$$\Pr\{K = k\} = \binom{m-1}{k} p^k q^{m-1-k}. \tag{3}$$

In the unbiased case, $p = q = \frac{1}{2}$, this model has been analysed by Knuth, Coffman and Eve, as well as Konheim and Newman; see the references in [9]. It is known for instance that the expected path length, under the biased model, is of the form

$$m \log_2 m + m \left(\frac{\gamma - 1}{\log 2} + \frac{3}{2} - \alpha + \delta(\log_2 m) \right) + O(m).$$

There $\alpha := \sum_{k\geq 1}(2^k-1)^{-1}$ and δ is a periodic function with magnitude less than 10^{-6}. Height and other parameters are studied by Aldous and Shields in [1].

Data compression. Consider an execution of the LZ78 algorithm when n characters of the text have been scanned, assuming that a new segment starts at position $(n+1)$. Under the binary memoryless model, the number of segments formed so far is a random variable M_n that is also the number of internal nodes of the companion dst built by the algorithm. By design, the dst is built of keys that obey Eq. (3), with $M_n = m$. It is also realized easily that the internal path length of the corresponding tree is equal to n.

Thus, there are two closely related models that are in a way "inverse" of each other:

- the *random dst model* where the number of strings m is fixed, and path length of the tree is to be analysed;
- the *compression model* where a dst is built by successive insertions until path length attains a fixed bound n, and the size of the tree (corresponding to the size of the compressed text) is to be analysed.

Analysis starts with the random dst model. Inversion will then be realized by an application of renewal theory.

3. The random digital search tree model

From an analytical standpoint, this model is the most natural one, since the random tree process given by (3) is a decomposable one. The novelty comes from the fact that, till recently, only the unbiased case $p = q = \frac{1}{2}$ had been analyzed. Jacquet and Szpankowski [7] have proved the following.

Theorem 1. *Let L_m be the path length of a digital search tree built on m random strings according to the Bernoulli model. Then, the mean and variance of L_m satisfy*

$$\mathrm{E}\, L_m = \frac{m}{h}\left(\log_2 m + \frac{h_2}{2h} + \gamma - 1 - \alpha + \delta_0(\log m)\right) + O(\log m),$$

$$\mathrm{Var}\, L_m = c_2 m \log m + O(m),$$

with δ_0 a periodic function of mean value 0 and small amplitude, and

$$h = p\log_2\frac{1}{p} + q\log_2\frac{1}{q},\ h_2 = p\log_2^2 p + q\log_2^2 q,\ \alpha = -\sum_{k=1}^{\infty}\frac{p^{k+1}\log_2 p + q^{q+1}\log_2 q}{1-p^{k+1}-q^{k+1}},\ c_2 = \frac{h_2 - h^2}{h^3}.$$

Furthermore, the distribution of L_m is asymptotically normal,

$$\frac{L_m - \mathrm{E}\, L_m}{\sqrt{\mathrm{Var}\, L_m}} \xrightarrow{d} \mathcal{N}(0,1).$$

As is well-known, there are three ways to conduct such an analysis, with Poisson, exponential, or ordinary generating functions (PGF, EGF, OGF). The proof given in [7] is of the Poisson type. The bivariate PGF satisfies then the nonlinear difference-differential equation,

$$\frac{\partial L(z,u)}{\partial z} = L(pzu,u)L(qzu,u),$$

with $L(z,0) = 1$. The approach starts with a quasi-linearization technique of [5], by considering $\ell(z,u) = \log L(z,u)$. By bootstrapping, the functional equation is solved in larger and larger pseudo-cones. Solutions are estimated asymptotically by means of the Mellin transform technology summarized in [3]. Then, mean, variance, and limit distribution follow for the Poisson model.

Translation to the Bernoulli model (*i.e.*, the binary memoryless channel) is then achieved by "analytic depoissonization", *i.e.*, an adequate use of the saddle-point method originating in Jacquet and Régnier's analysis of path length in tries. The technical difficulties are rather formidable, but the results obtained are extremely precise.

4. **The compression model**

It is now possible to return to the LZ78 algorithm. We recall that M_n is the number of segments (or phrases) built on a random text of length n. We have:

Theorem 2. *Let M_n be the number of phrases built on n characters of the text by LZ78, according to the binary memoryless model. Then, the mean and variance of M_n satisfy*

$$\mathrm{E}\, M_n \sim \frac{nh}{\log_2 n}, \qquad \mathrm{Var}\, M_n = \frac{c_2 h^3 n}{\log_2^2 n}.$$

Furthermore, the distribution of M_n is asymptotically normal,

$$\frac{M_n - \mathrm{E}\, M_n}{\sqrt{\mathrm{Var}\, M_n}} \xrightarrow{d} \mathcal{N}(0,1).$$

The proof of this result is in [7]. The ideas have been further refined by Louchard and Szpankowski in [8], and this leads to a second order asymptotic analysis, hence a characterization of redundancy of LZ78.

The "inverse" relations alluded to in Section 2 are expressed precisely by the equality,

$$\Pr\{M_n > m\} = \Pr\{L_m \le n\}.$$

This is known as the *renewal equation*. Theorem 1 gives good estimates of the right hand side. Roughly speaking, we have, by Theorem 1, a known dependency between L (path length) and M (size of the dst),

$$L_M \approx \frac{1}{h} M \log_2 M + \sqrt{c_2 M \log_2 M}\, X,$$

where X is a unit Gaussian variate that represents random fluctuations. This can be formally inverted, leading to

$$M \approx \frac{Lh}{\log_2 L} + \sqrt{\frac{c_2 h^3 L}{\log_2^2 L}}\, X.$$

This is exactly what Theorem 2 expresses and the process is made valid by an appeal to the general theory of renewal equations; see [2, Thm. 17.3].

From there, it is possible to solve the redundancy problem of Section 1; see [8] for details. Define the redundancy of LZ78 as

$$\overline{r}_n = \frac{1}{n} \left(\mathrm{E}\{M_n(\log_2 M_n + 1)\} - nh \right).$$

The idea is that there are M_n phrases and each of them costs about $\log_2 M_n$ bits. (The term $+1$, there, is implementation specific but not essential.)

Then moment bounds and inequalities justify a precise version of the approximation

$$\mathrm{E}\{M_n \log_2 M_n\} \approx \mathrm{E}\{M_n\} \log_2 \mathrm{E}\{M_n\}.$$

Also, the speed of convergence to the normal limit in Thm. 1 is seen to be $O(m^{-1/2})$. This gives all the ingredients for a second-order asymptotic analysis of LZ78.

Theorem 3. *The global redundancy of Lempel-Ziv's LZ78 algorithm is* $O(1/\log n)$. *More precisely,*

$$\overline{r}_n \sim \frac{1}{\log_2 n}\left(2h - h\gamma - \frac{1}{2}h_2 + h\alpha - h\delta_0(\log_2 n)\right).$$

Note that the corresponding redundancy of LZ77 is known to be

$$\overline{r}^*_n = O(\frac{\log\log n}{\log n}),$$

and this is conjectured to be the right order. Therefore, assuming this conjecture, *the LZ78 algorithm—based on digital search trees and segment boundaries—is less "redundant" and deviates less than LZ77 from the info*