

Of Lempel-Ziv-Welch Parses with Refillable Gaps

Alberto Apostolico*

Universit  di Padova & Purdue University

1 Introduction and Summary

We consider gapped variants of classical data compression paradigms by Ziv, Lempel, and Welch [12]. In the original algorithm, *phrases* are identified and stored in a *dictionary* on-the-fly as the textfile is scanned. The entries in the dictionary are then matched against the incoming string, thereby determining the next codeword, and this gives the method an inherently linear-time implementation. In our variants, the phrases used in compression are selected among suitably chosen strings of intermitently solid and wild characters produced by the autocorrelation of the sourcestring, in a way that still preserves linearity of time. At the receiver, gaps can be filled back exactly, interpolated, or left blank, so that lossless as well as lossy implementations are possible. However, the focus of this paper is on lossy variants.

Lossy compression by textual substitution has been variously defined and studied in recent years. We refer to, e.g., [6] for a recent survey of motivation and results. Most of the lossy extensions of ZL schemata proceed by finding a best match, within an assigned fidelity, between the incoming stream and the already seen portion of the text or some suitably pre-arranged external dictionary. Irrespective of the criterion used to find such an optimum match, the search itself is inherently time consuming. From the standpoint of computation, the reason for this resides ultimately with the fact, that pairwise matching of symbols is transitive while mis-matching is not. One more difficulty may be introduced by the need to preserve the information relative to the positions of mismatches in a phrase, which poses extra overhead in encoding. The state of the art and challenges ahead are nicely captured in [6]:

“All universal lossy coding schemes found to date lack the relative simplicity that imbues Lempel-Ziv codes and arithmetic codes with economic viability. Perhaps as a consequence of the fact that approximate matches abound whereas exact matches are unique, it is inherently much faster to look for an exact match than it is to search from a plethora of approximate matches looking for the best, or even nearly the best,

*Dipartimento di Ingegneria dell' Informazione, Universit  di Padova, Padova, Italy and Department of Computer Sciences, Purdue University, Computer Sciences Building, West Lafayette, IN 47907, USA. Work Supported in part by the Italian Ministry of University and Research under the National Projects FIRB RBNE01KNFP, and PRIN “Combinatorial and Algorithmic Methods for Pattern Discovery in Biosequences”, and by the Research Program of the University of Padova. axa@dei.unipd.it

among them. The right way to trade off search effort in a poorly understood environment against the degree to which the product of the search possesses desired criteria has long been a human enigma. This suggests it is unlikely that the “holy grail” of implementable universal lossy source coding will be discovered soon.”

The emphasis in this paper is on speed of encoding and decoding, which will rest in particular on an effective, implicitly encodable choice of the gap positions in a phrase. The family of methods considered is inspired by the notion of a *motif*. In a motif driven LZW parse, dictionary entries correspond to strings written on the alphabet $\Sigma \cup \{-\}$, where “-” represents a wildcard or “don’t care” symbol that may take up one of several specifications. Allowing don’t cares leads to an increase of the average length of phrases hence to a decrease in their number. As the tables reported in [2, 3, 5] display, savings can be dramatic for inputs such as images and signals, in which the gaps can be filled by interpolation at the receiver with negligible loss. As it turns out, however, improvements are also seen in lossless schemata in which a dictionary of *resolvers* must be added to disambiguate the don’t cares in the phrases.

The phrases used in the present paper are a deterministically generated set of approximate patterns with don’t cares, that result from the self-correlation of the source-string. In so far as it rigidly dictates the structure of phrases, this rigidity is likely to affect the efficiency of the encoding; in exchange, it affords computation in linear time. As mentioned, the particular choice made is inspired by recently developed combinatorial notions and properties of motif patterns, that set the framework for picking don’t cares. We refer to [4] for a more formal treatment. Motif-based *off-line* algorithms were introduced and tested in [2, 3, 5]. The majority of those applications assume a preprocessing to extract motifs from the textstring. Here we concentrate on on-line algorithms.

The remainder of the paper is organized as follows. In the next section, we evoke a match length and recurrence time backstage for our algorithms. Next, we develop and analyze the algorithm of Figure 1 and some of its variants. Following that, we compute projected compressions in correspondence with extremal sequences. Some tables reporting from experiments are given after that, and a list of conclusions and plans for future work close the paper.

2 Match lengths

Assume that, in determining the next LZW phrase, we are willing to accept a few mismatches in exchange for a longer phrase. We consider the expected length of such a phrase assuming a maximum density d of mismatches per phrase. More formally, we will take the per-symbol *distortion* $d(w, \hat{w})$ for two strings of m characters to be the average Hamming Distance $d = (1/m) \sum_{i=1}^m d(w_i, \hat{w}_i)$ where $d(w_i, \hat{w}_i) = 1$ if $w_i \neq \hat{w}_i$ and $d(w_i, \hat{w}_i) = 0$ if $w_i = \hat{w}_i$. An average *fidelity* $f = 1 - d$ is naturally associated with a distortion of d .

For fixed d or f assuming a source that emits symbols from Σ with an iid distribution, consider two strings of m characters and set $p = \sum_{i=1}^{|\Sigma|} p_i^2$ the probability of a single match. Then, the probability of a match at distortion d or fidelity f between

the two strings is

$$B(m, p, f) = p^{(mf)}(1-p)^{(m-mf)} \binom{m}{mf} = p^{(mf)}(1-p)^{(md)} \frac{m!}{(md)!(mf)!}$$

Using Stirling approximation $\ln(n!) = n(\ln n - 1)$ on the logarithm of $B(m, p, f)$, and going back by exponentiation we get, after easy passages, $B(m, p, f) = e^{-m\mathcal{H}(f,p)}$ where $\mathcal{H}(f, p)$, defined as $\mathcal{H}(f, p) = f \log(f/p) + (1-f) \log(1-f)/(1-p)$ is the *relative entropy* or Kullback-Liebler distance between the f and p ‘coins’. $\mathcal{H}(f, p)$ is always positive, as a consequence of Gibbs theorem, well known to physicists, according to which for two distributions (actually, sets of real numbers of same sum) p_i and q_i , we have $-\sum_{i=1}^n p_i \log p_i \leq -\sum_{i=1}^n p_i \log q_i$.

We overestimate the length of a longest match at distortion d between the incoming phrase and those already in the dictionary as follows. For two random sequences $X_1, X_2, \dots, X_i, \dots$ and $Y_1, Y_2, \dots, Y_i, \dots$, let $C_i \equiv \mathcal{I}(X_i = Y_i)$ be defined to have value 1 if $X_i = Y_i$, ($i = 1, 2, \dots, j, \dots$) and 0 otherwise, and let

$$R_n^f = \max \left\{ m : fm \leq \sum_{1 \leq k \leq t} C_{i+k}, 0 \leq i \leq n - m \right\}.$$

By a known result by Erdős-Rényi, when the C_i 's are independent Bernoulli variables with probability p and $p < f \leq 1$, we have that $P(R_n^f / \log(n) \rightarrow 1/\mathcal{H}(f, p)) = 1$. In intuitive terms, this can be derived by taking $e^{-m\mathcal{H}(f,p)}$ as the probability of a dense headrun of length m , and forcing such a headrun to occur with certainty at any one of about n starting positions. Thus, setting $l = ne^{-m\mathcal{H}(f,p)}$, we obtain $R_n^f = m = \log(n)/\mathcal{H}(f, p)$. Note the analogy of the above with the Asymptotic Equipartition Property for an i.i.d. random source, which states that the logarithm of the inverse of the probability (which is, by Kac's Lemma, the expected logarithm of the recurrence time) of a string, divided by its length, is close to the entropy [7,15].

It is easy to see that forfeiting the search for the *longest* match has the undesired effect of bringing the size of the expected match length down to a constant. With $p = \sum_{i=1}^{|S|} p_i^2$ the probability of a single match, the probability of a match of length exactly m is $p^m(1-p)$. The corresponding average value or expected length is $\sum mp^m(1-p)$ and is computed according to the formula $\sum kx^k = \frac{x}{(1-x)^2}$ for $|x| < 1$. Hence the expected length is the well known $p/(1-p)$

Allowing a density of d for don't cares, the probability of one match becomes $[d + (1-d)p]$, whence that of exactly m matches becomes $\hat{P} = [d + (1-d)p]^m(1-d)(1-p) = [1 - (1-p)(1-d)]^m(1-d)(1-p)$.

We can write

$$\hat{P} = (1-d)(1-p) \sum_{k=0}^m \binom{m}{k} [(p-1)(1-d)]^k.$$

Note that since $x = d + (1-d)p$ is a probability (the probability of a single match) we must have $0 \leq x \leq 1$ hence $-1 \leq x-1 = (p-1)(1-d) = (d-1)(1-p) \leq 0$. If we

stipulate that $|x| < 1$, then we can use the fact that for any (integer or non-integer) m

$$\textcircled{=} = (1-x)(1+x)^m = (1-x) \sum_{k=0}^m \binom{m}{k} x^k = \sum_{k=0}^{\infty} \binom{m}{k} x^k.$$

If now, in

$$\sum kx^k = \frac{x}{(1-x)^2}$$

we set $x = d + (1-d)p$ we get the new expected match length

$$\hat{m} = \frac{d + (1-d)p}{(1-p)(1-d)} = \frac{1}{(1-p)(1-d)} - 1.$$

Thus the ratio of the average match allowing a distortion of d over that of no distortion is

$$\frac{\hat{m}}{m} = \frac{d + (1-d)p}{(1-p)(1-d)} \cdot \frac{(1-p)}{p} = \frac{d + (1-d)p}{p(1-d)} = 1 + \frac{d}{p(1-d)}$$

The discussion of this section seems to suggest that finding a *longest* match is crucial to parsing schemata that aimed at asymptotic entropy rate. However, one key factor that presides over the performance of ZL methods is the *distinctness* of phrases in a parse [7, 9]. In the next section, we introduce a mechanism, patterned after LZW, for producing in linear time both lossy and lossless parses consisting of distinct phrases which deterministically allocate gaps and the distortion that goes with them.

3 Motif Driven ZL Compression

The generic stage of LZW may be considered as consisting of two parts, as follows. In the first part (hereafter, *seek phrase*), a longest matching phrase from the dictionary is found, and its index is appended to the output. In the second, the one-symbol extension of the *current occurrence* of the phrase is added to the dictionary for possible future reference. In our adaptation, Part 1 is identical, except that the output now must contain both the reference to the phrase and also the characters for its disambiguation. This is achieved through the use of two dictionaries that grow hand-in-hand, whereby a phrase is resolved in the shuffle of a word from the main dictionary and one from the auxiliary one. Looking for a *best* phrase, e.g., the one minimizing mismatches is feasible but time consuming, hence our implementation of seek-phrase greedily pursues matches over mismatches. If we assume $\Sigma \cup \{-\}$ to be sorted with $\{-\}$ its maximum element, then seek-phrase finds the lexicographically least phrase occurring at the current position.

The pseudo-code of Fig. 1 describes the algorithm. At the outset, each phrase in the parse is decomposed in a pair consisting of a primary phrase s over $C \cup \{-\}$ and an auxiliary resolver s' over C . A suitable shuffle of s and s' reconstructs the actual phrase over C . The information needed for the shuffle is the set of positions of s occupied by don't cares. However, this does not need to be supplied explicitly.

```

Initializations
Initialize dictionary trie and resolver trie with the characters from  $\Sigma$ 
Initialize phrase  $s$  to first input character, resolver  $s'$  to the empty word  $\lambda$ 
Set output  $\leftarrow \langle code(s), code(s') \rangle$ 
Body Repeat until no more input characters
   $\sigma \leftarrow$  read the next input character
  if  $s\sigma$  is in the dictionary then set  $s = s\sigma$  (seek-phrase continues)
  else, if  $s_.$  is in the dictionary and  $s'\sigma$  is in resolvers
    then set  $s = s_.$ ,  $s' = s'\sigma$  (seek-phrase continues)
  else (the end of a stored phrase has been reached)
    1- output  $\leftarrow$  output  $\cdot \langle code(s), code(s') \rangle$ 
    2- if  $s\sigma'$  for some  $\sigma' \neq \sigma$  is in the dictionary then
      2a- add  $s_.$  to dictionary
      2b- if  $s'\sigma$  is not already in it add  $s'\sigma$  to resolvers
      else add  $s\sigma$  to dictionary
    3-  $s \leftarrow \sigma$ 
end Body

```

Figure 1: LZWA: a motif-driven LZW

Rather, the gaps are filled with the characters from s' , in exact succession, and the mechanics of the encoding and decoding, respectively, takes care of consistency.

Theorem 1 *Algorithm LZWA works in time and space linear in the source text x .*

Proof: Inherited straightforwardly from LZW. ■

Theorem 2 *Except possibly for the root, the dictionary trie built by LZWA is a binary tree, in which each internal node has either one node labeled by some character from Σ or two nodes, one labeled by a character from Σ and one by the don't care “_”.*

Proof: We discuss the operation and terminating condition for LZWA. Let \bar{x} be the suffix of the source yet to be encoded. The search for the new phrase can be segmented into three main parts. In Part 1, we seek the longest path in the trie that matches a prefix of \bar{x} . This part terminates in one of two possible ways, which we call Mode A and B, respectively, depending on whether the last attempted transition following s reached a leaf or an internal node. In the first case, all conditions must fail so that the last assignment under 2 is reached and the dictionary trie is extended with a single new arc labeled by a , hence by a solid character. In Mode B, s ends at an internal node, call it ν , and we cannot have simultaneously that $s_.$ is in the dictionary and $s'\sigma$ is in the resolvers. If the cause for termination is that $s_.$ is not in the dictionary, then inductively ν has one child and the corresponding arc must be labeled by some $a' \neq a$: the algorithm adds an arc labeled “_” to ν , which makes ν a binary node, and predisposes a resolver $s'\sigma$ for the future. On the other hand, if the default is on $s'\sigma$ not being in the resolvers, then inductively ν has already two children, and the action of the algorithm is limited to adding $s'\sigma$ to the resolvers. ■

We leave it as an exercise to show that correct decoding is possible both for the lossless as well as the lossy encoding, and it works in linear time. It is worth to pinpoint a few modifications and upgrades of LZWA that have no substantial bearing on time performance. For instance, a *cheaper encoding of resolvers* results from encoding only the leaves of the aux trie, since the longest extension of a resolver is enough to drive decoding. Also, *error density bounds* where a maximum number k of errors is allowed in each phrase may be introduced. Further, one may *force each phrase to be a consensus* by making sure that every phrase used in the encoding is a substring of the consensus of two suffixes of the source. One may develop *h-ary dictionary tries* with $h > 2$, may be developed, by issuing a don't care transition not the second, but the h-th time a same dictionary node is reached. Finally, in lossy compression, the condition of *simultaneous matching on resolver trie* may be forfeited as an undesirable bottleneck.

4 Comparing Vocabulary Build-ups

The tradeoff between LZW and the lossless LZWA alternates in experiments. On the other hand, a theoretical comparison of the compression performances associated with LZW and LZWA parses seems not trivial. The analysis below attempts at capturing the mechanics of the savings, but first note that even the lossy (i.e., with resolvers discarded) version of LZWA does not always save over LZW. For this, consider the extreme case of a string a^n formed by n identical symbols. The number of distinct substrings in such a string is only n . The string achieves a better compression under LZW, and this also shows that the transition to LZWA is not advantageous for all inputs. In fact, a^n is parsed in almost the same way by LZW and LZWA, roughly producing a number of phrases in the parse equal to $n = \sum_{i=1}^t i$, i.e., $n \approx t^2/2$ whence $t \approx \sqrt{2n}$ which corresponds to an encoding $t \log t \approx 1/2\sqrt{2n} \log n = 0.71\sqrt{n} \log n$. LZWA does not introduce any extra phrases in the dictionary, and yet it brings about an overhead on the encoding which must now take into account the symbol “-”. This costs at least one extra bit.

The situation is different when LZW and LZWA are compared on their respective most incompressible input. We conform to some of the notation and treatment in [9, 13, 14]. With $|\Sigma| = a$, consider a string formed by the sequence of all words of length $1, 2, 3, \dots, k$, in lexicographic order. The length of such a string is

$$n_\alpha(k) = \sum_{i=1}^k i\alpha^i = \frac{\alpha}{\alpha-1} \left[\alpha^k \left(k - \frac{1}{\alpha-1} \right) + \frac{1}{\alpha-1} \right]$$

which for $a = 2$ becomes, in particular $n_2(k) = (k-1)2^{k+1} + 2$, or, approximately, $n_2(k) \approx k2^{k+1}$, and for a large enough such that $a \approx a-1$ becomes $n_\alpha(k) \approx k\alpha^k$. For $a = 3$, we have

$$n_3(k) = \frac{3}{2} \left[3^k \left(k - \frac{1}{3} \right) + \frac{1}{2} \right] = \frac{1}{2} \left(k3^{k+1} - 3^k + \frac{3}{2} \right)$$

whence, for $k > 0$, $1/2k3^k < n_3(k) < 1/2k3^{k+1}$. Thus, we can state in general that as a increases from small values to large values $n_\alpha(k)$ goes from $\Theta(k\alpha^{k+1})$ to $\Theta(k\alpha^k)$.

k	$n_{10}(k)$	$N_{10}(k)$	$N_2(\bar{k})$	$n_2(k)$	\bar{k}	$n_2(\bar{k} + 1)$
1	10	10	4	2	1	10
2	210	100	16	98	4	258
3	3210	1,000	256	1538	7	3586
4	43210	10,000	4,096	40962	11	90114
5	543210	100,000	32,768	425986	14	917506
6	6.54321e+006	10^6	262,144	4.19431e+006	17	8.9129e+006
7	7.65432e+007	10^6	2,097,152	3.98459e+007	20	8.38861e+007
8	8.76543e+008	10^6	33,554,432	7.71752e+008	24	1.61061e+009
9	9.87654e+009	10^6	268,435,456	6.97932e+009	27	1.44955e+010
10	1.09877e+011	10^{10}	$2,147x10^6$	6.2277e+010	30	1.28849e+011
15	1.65432e+016	10^{15}	$281x10^{12}$	1.29478e+016	47	2.64586e+016
20	2.20988e+021	10^{20}	$18x10^{18}$	1.1437e+021	63	2.32429e+021
40	4.4321e+041	10^{40}	$232x10^{37}$	3.51171e+041	130	7.07787e+041

Table 1: Phrase vocabulary growth for $\alpha = 10$ and $\alpha = 2$.

The number of distinct substrings in any string of the type considered is the maximum possible for that alphabet and length. This number is:

$$N_\alpha(k) = \sum_{i=1}^k \alpha^i = \frac{\alpha}{\alpha - 1} (\alpha^k - 1)$$

in general, which becomes approximately α^k for large alphabets and $2^{k+1} - 2 \approx 2^{k+1}$ for the binary alphabet. Thus, we can state that in general the vocabulary size $N_\alpha(k)$ of our string goes from $\Theta(\alpha^{k+1})$ to $\Theta(\alpha^k)$ in the transition from a very small alphabet to a very large one.

We may regard the introduction of don't cares as a means of collapsing the alphabet of the main trie, in the sense that once the current phrase is found to diverge from its path in the trie, this creates a don't care transition that will allow any other phrase reaching that junction in the future to expand into a longer match. Consequently, the expectation is that x is partitioned into longer phrases and there will be correspondingly fewer of them. In lossless implementations, this gain seems offset by the increased cost of phrase encoding.

In lossy schemes, we forfeit resolvers, and the question becomes natural as to what savings on the number of phrases is brought about by this collapse of the alphabet. This kind of analysis seems not easy in general. However, one rough estimate is offered by the comparison of the values taken by $N(k)$ for comparable input lengths but under different alphabet sizes. As an example, the table summarizes the phrase dictionary sizes (approximated by 10^k and $2^{\bar{k}+1}$, respectively) that correspond to strings of approximately the same length, for decimal and binary alphabets.

Analytical formulas are harder to come by. We begin by recalling the notion of a d -ary tree, which is defined recursively as consisting of the empty tree or a root node having up to d d -ary trees as its children. Given a d -ary tree T , its *extension* is obtained by adding *leaves* in such a way that every node originally in T has now exactly d children. The following property is easily checked.

Lemma 1 An extended d -ary tree with m internal nodes has precisely $(d - 1)m + 1$ leaves.

Further, let the level of a node to be 0 for the root and 1 plus the level of the father otherwise. Now, define the external (respectively, internal) path length of an extended d -ary tree as the sum of the levels of all leaves (resp., internal nodes), and denote them by $E(T)$ and $I(T)$, respectively. It is easy to verify that:

Lemma 2 In any *extended* d -ary tree, $E(T) = (d - 1)I(T) + d \cdot m$

Finally, we recall the following property of extended d -ary trees (see, e.g., [8,10]).

Theorem 3 With $\ell = \lceil \log_d[(d - 1)m + 1] \rceil$, the minimum external path length for a d -ary tree of m nodes is

$$\lceil m(d - 1) + 1 \rceil \ell - \frac{d^{\ell+1} - d}{d - 1} + d \cdot m.$$

Combining Theorem 3 with Lemma 2 we get the expression

$$\min(I) = \frac{\lceil m(d - 1) + 1 \rceil!}{(d - 1)} + \frac{d^{\ell+1} - d}{(d - 1)^2}.$$

for the minimum internal path length.

The import of the above derivation to our context is as follows. In a string of length n created by the concatenation of all distinct words of length up to k as above, we have that the number $N(k)$ of distinct phrases corresponds to the number m of internal nodes, and the length $n(k)$ of the string itself is the internal path length in a tree of minimum external path length, i.e., $N(k) = m$ and $n(k) = \min(I)$. This gives us a handle to compare the maximum number of distinct phrases that can be packed in n positions using alphabets of different cardinality, say, a large alphabet of size α and a small one of size β .

Upon approximating ℓ to $\log_d(d - 1) + \log_d m$ we have

$$\begin{aligned} \min(I) &\approx \frac{\lceil m(d - 1) + 1 \rceil (\log_d(d - 1) + \log_d m)}{(d - 1)} + \frac{d(d - 1)m}{(d - 1)^2} \\ &= \frac{\lceil m(d - 1) + 1 \rceil (\log_d(d - 1) + \log_d m) + dm}{(d - 1)} \\ &= m[\log_d m + \log_d(d - 1)] + \frac{1}{(d - 1)} (\log_d(d - 1) + \log_d m + md). \end{aligned}$$

We see that for large $d = \alpha$ the logarithm $\log_d(d - 1)$ goes to 1 and so does $d/(d - 1)$, and we get

$$\min_{\alpha}(I) = m \log_{\alpha} m + 2m + o(m).$$

For small $d = \beta$, that logarithm tends to 0 but $d/(d-1)$ approaches 2, thus we also get $\min_{\beta}(I) = m \log_{\beta} m + 2m + o(m)$. We may now force $\min_{\alpha}(I) = \min_{\beta}(I)$ and derive the relationship between the largest vocabularies achievable while parsing two strings of equal length but written, respectively, over a large and a small alphabet. When the length n of the input string tends to infinity so does m , and for $m \rightarrow \infty$ we may neglect smaller order terms and impose: $n = \min_{\alpha}(I) = N_{\alpha} \log_{\alpha} N_{\alpha} + 2N_{\alpha} = N_{\beta} \log_{\beta} N_{\beta} + 2N_{\beta} = \min_{\beta}(I) = n_{\beta}$, that is, $N_{\alpha} \log_{\alpha} N_{\alpha} + 2N_{\alpha} = N_{\beta} \log_{\beta} \alpha \log_{\alpha} N_{\beta} + 2N_{\beta}$.

Theorem 4 As $n \rightarrow \infty$, for $a \geq \beta^2$, it is $N_{\beta} < N_{\alpha} < N_{\beta} \log_{\beta} a$.

Proof: Irrespective of the sign of $(N_{\alpha} - N_{\beta})$, we have

$$\begin{aligned} N_{\beta} \log_{\alpha} N_{\beta} &= \frac{N_{\alpha} \log_{\alpha} N_{\alpha} + 2(N_{\alpha} - N_{\beta})}{\log_{\beta} \alpha} < \frac{N_{\alpha} \log_{\alpha} N_{\alpha} + 2N_{\alpha}}{\log_{\beta} \alpha} < \\ &\frac{1}{\log_{\beta} \alpha} [N_{\alpha} \log_{\alpha} (\frac{N_{\alpha}}{\log_{\beta} \alpha}) + 2N_{\alpha} + N_{\alpha} \log_{\alpha} \log_{\beta} \alpha] < \\ &\frac{N_{\alpha}}{\log_{\beta} \alpha} (\log_{\alpha} (\frac{N_{\alpha}}{\log_{\beta} \alpha}) + 3) < \frac{2N_{\alpha}}{\log_{\beta} \alpha} \log_{\alpha} (\frac{2N_{\alpha}}{\log_{\beta} \alpha}), \end{aligned}$$

since $\log_{\beta} a < a$ and for $n \rightarrow \infty$ we have $\log_{\alpha} N_{\alpha} \geq 4$, whence

$$N_{\beta} < \frac{2N_{\alpha}}{\log_{\beta} \alpha} \quad \text{or} \quad N_{\alpha} > \frac{N_{\beta}}{2} \log_{\beta} \alpha$$

and $(N_{\alpha} - N_{\beta})$ is positive for $\log_{\beta} \alpha \geq 2$ or $a \geq \beta^2$. Under the same conditions, we have that

$$N_{\beta} \log_{\alpha} N_{\beta} = \frac{N_{\alpha} \log_{\alpha} N_{\alpha} + 2(N_{\alpha} - N_{\beta})}{\log_{\beta} \alpha} > \frac{N_{\alpha}}{\log_{\beta} \alpha} \log_{\alpha} N_{\alpha} > \frac{N_{\alpha}}{\log_{\beta} \alpha} \log_{\alpha} (\frac{N_{\alpha}}{\log_{\beta} \alpha})$$

whence $N_{\alpha} < N_{\beta} \log_{\beta} \alpha$. ■

Taking the ratio of the encodings of the two strings using $t \log t$ as the number of bits that are needed to encode t phrases, we get:

$$\frac{N_{\beta} \log N_{\beta}}{N_{\alpha} \log N_{\alpha}} > \frac{N_{\beta} \log N_{\beta}}{N_{\beta} \log_{\beta} \alpha (\log N_{\beta} + \log \log_{\beta} \alpha)} = \frac{1}{\log_{\beta} \alpha + \frac{\log \alpha \log \log_{\beta} \alpha}{\log N_{\beta}}}.$$

Conclusions and Plans for Future Work

Most previous lossy variants of ZL and related family of encoders are built around the iterated quest for best matches within an assigned fidelity. This results in algorithms that are inherently superlinear and not easy to implement and analyze. The approach followed in this paper concentrates on time performance, and builds a parse in which phrases are all distinct, with a structure that is based on self-correlations of the source

and dictated deterministically by the very process of parsing. The scheme is easily implemented in linear time. The analysis of performance of lossy compression by textual substitution is not easy in general and the case arising in this paper is no exception. In practice, however, companion schemata of off-line lossy and lossless motif based compression and grammatical inference for documents of various nature have been successfully tested previously [2, 3, 5], and the compression achieved by LZWA and its variants are encouraging. Further variants and deeper analyses seem thus worthy of pursuit.

Acknowledgements: I am indebted to M. Comin, M. Melucci, M. Regnier, J. Storer, J. Ziv and M. Ziv-Ukelson for discussions and help.

References

- [1] A. Apostolico and Z. Galil (Eds.), *Pattern Matching Algorithms*, Oxford University Press, New York (1997).
- [2] A. Apostolico, M. Comin and L. Parida, "Bridging Lossy and Lossless Data Compression by Motif Pattern Discovery", in *General Theory of Information Transfer and Combinatorics*, Vol. II of a Report on a Research Project at the ZIF (Center of interdisciplinary studies) in Bielefeld Oct. 1, 2002 – August 31, 2003, edited by R. Ahlswede with the assistance of L. Baumer and N. Cai - in press (2004).
- [3] A. Apostolico, M. Comin and L. Parida, "Motifs in Ziv-Lempel-Welch Clef" Proceedings of *IEEE DCC Data Compression Conference*, pp. 72–81 Computer Society Press, (2004).
- [4] A. Apostolico and L. Parida, "Incremental Paradigms of Motif Discovery", (2001) *Journal of Computational Biology*, 11:1, 15–25 (2004).
- [5] A. Apostolico and L. Parida, "Compression and the Wheel of Fortune", *Proceedings of DCC 2003*, IEEE Computer Society Press, 143–152 (2003).
- [6] T. Berger and J.D. Gibson, "Lossy Source Coding," *IEEE Trans. on Inform. Theory*, vol. 44, No. 6, pp. 2693–2723, 1998.
- [7] T. M. Cover and J. A. Thomas, *Elements of Information Theory*, Wiley-Interscience (1991).
- [8] D.E. Knuth, *The Art of Computer Programming - Vol. 1*, Addison-Wesley, Reading, Mass. (1968).
- [9] A. Lempel and J. Ziv, "On the complexity of finite sequences," *IEEE Trans. on Inform. Theory*, vol. 22, pp. 75–81, 1976.
- [10] Reingold, E.M., J. Nievergelt and N. Deo, *Combinatorial Algorithms: Theory and Practice*, Prentice-Hall, Englewood Cliffs, (1977).
- [11] J. A. Storer, *Data Compression: Methods and Theory*, Computer Science Press (1988).
- [12] T. A. Welch, "A Technique for High-Performance Data Compression", *IEEE Computer* 17:6, 8–19, (1984).
- [13] J. Ziv and A. Lempel, "A universal algorithm for sequential data compression," *IEEE Trans. on Inform. Theory*, vol. IT-23, no. 3, pp. 337–343, (1977).
- [14] J. Ziv and A. Lempel, "Compression of individual sequences via variable-rate coding," *IEEE Trans. on Inform. Theory*, vol. 24, no. 5, pp 530–536 (1978).
- [15] A.D. Wyner, J. Ziv and A.J. Wyner, "On the Role of Pattern matching in Information Theory," *IEEE Trans. on Inform. Theory*, vol. 44, no. 6, pp 2045–2056 (1998).