On the Average Redundancy Rate of the Lempel-Ziv Code with the K-Error Protocol

Yuriy A. Reznik [a,1], Wojciech Szpankowski [b,2,3]

[a] *RealNetworks, Inc., 2601 Elliott Avenue, Seattle, WA 98121*

[b] *Department of Computer Science, Purdue University, W. Lafayette, IN 47907*

Abstract

In this paper we examine the average redundancy rate of a *Lempel-Ziv'78* code with the *k-error protocol*. Storer and Reif have studied this modification of the *Lempel-Ziv* scheme and showed that it provides an efficient protection against error propagation while preserving the asymptotic optimality of the code. We refine this result by providing an asymptotic expression for the *average redundancy rate* of this code for memoryless sources. Using analytic methods, we establish our result by exploiting a relationship between a parsing scheme of the *Lempel-Ziv* encoder with the *k-error protocol* and a generalization of the *digital search tree structure*. We accompany our analysis with a number of experiments that test the validity of our theoretical result and demonstrate the effects of various additional modifications of the *Lempel-Ziv* algorithm.

Key words: Average redundancy rate, *Lempel-Ziv parsing scheme*, data compression, *generalized Lempel-Ziv scheme*, *digital search trees*, asymptotic analysis, error propagation, *k-error protocol*

1 Introduction

The essential component of several universal data compression algorithms is a parsing scheme due to Ziv and Lempel [19]. It is a dictionary-based algorithm that partitions a sequence into phrases (blocks) of variable sizes such that a new block is the shortest sub-string not seen in the past as a phrase. A new

[1] This author is also on leave from the Institute of Mathematical Machines and Systems of the National Academy of Sciences of Ukraine.

[2] Corresponding author. E-mail: spa@cs.purdue.edu

[3] This author was supported by NSF grants: NCR-9415491 and C-CR-9804760.

phrase is coded by giving the location of a prefix (that occurred before as a phrase) and the value of the last symbol. That is, the *Lempel-Ziv* (or LZ) code consists of pairs in the format *(pointer, symbol)* [1,16]. For example, a sequence *ababbababaaaaaaaaac* is parsed into *(a)(b)(ab)(ba)(bab)(aa)(aaa)(aaaa)(c)* and its LZ code becomes *0a0b1b2a4b1a6a7a0c* (e.g., a pair *6a* indicates that this phrase consists of the sixth phrase as a prefix that occurred before being appended by a). The decoding is straightforward: read *(pointer, symbol)* pairs, construct a new phrase, and add a pointer to it in the dictionary.

One of the important drawbacks of the LZ code is its susceptibility to error propagation. That is, a single error on the communication channel (such as insertion, deletion, or change of an index) could cause the dictionaries of the encoder and decoder to differ, which in the worst case could corrupt all data that follows. E.g. if in our previous example the sixth pair *(1a)* was replaced with *(5a)* the result of the decompression would be: *ababbabab[babababaabab]aaac* (we have surrounded the corrupted part of the sequence by square brackets *[.]*).

To somewhat remedy this situation, Storer and Reif [17] have proposed a modification to the LZ algorithm that ensures that the encoder and decoder have identical dictionaries if the number of transmission errors does not exceed some fixed constant k. This method, also called *k-error protocol*, alters the original LZ scheme by the following steps [17]:

- Instead of assigning a new string to the next available location, the *k-error protocol* hashes *(pointer, symbol)* pairs into the entire address space of the dictionary.
- Counts are maintained for every *(pointer, symbol)* pair processed thus far, and each location in the dictionary is used for further references only if there is a clear winner among all *(pointer, symbol)* pairs hashed to this location (i.e. there is a phrase such that its count surpasses counts of the other *(pointer, symbol)* pairs hashed to the same address by some constant $b \geqslant 1$).

For instance, assuming that hashing is perfect (and thus each (pointer, symbol) pair is mapped into a unique address), and that the constant $b = 3$, a sequence *ababbababaaaaaaaaac*, will be parsed into *(a)(b)(a)(b)(b)(a)(ba)(ba)(aa)(aa)* *(aa)(aac)*. If we denote a hash value of a pair *(p,s)* as $\{p,s\}$, we can write the modified LZ code for the above sequence as: $\{0,a\}\{0,b\}\{0,a\}\{0,b\}\{0,b\}\{0,a\}$ $\{\{0,b\},a\}\{\{0,b\},a\}\{\{0,a\},a\}\{\{0,a\},a\}\{\{0,a\},a\}\{\{\{0,a\},a\},c\}$. Now, if we replace a 9-th index in this code $\{\{0,a\},a\}$ with the hash value of another valid string $\{\{0,b\},a\}$, the decoder will indeed decode this string incorrectly (i.e. we will have *(ba)* instead of *(aa)*), however, since the reference to *(aa)* is going to be transmitted 2 more times ($b = 3$), the decoder will be able to reconstruct the rest of sequence without any further errors. Thus, it will produce: *ababbababa[b]aaaaaaac* (*[.]* shows the corrupted part of the sequence).

It is important to note that the *k-error protocol* does not provide a perfect protection against changes in the compressed stream. It only ensures that a limited number of errors will not propagate. Thus in the above example, we were able to limit the effect of a single error to only one phrase in the decompressed sequence. This raises an interesting question about the value parameter b should have in order to prevent propagation of up to k errors. The answer provided by Storer and Reif [17, *Lag theorem*] is that for any combination of k errors (such as insertion, deletion, and change of an index sent over the channel) it is sufficient to select $b \geqslant 3k$. If a channel has only change-type errors, this bound can be adjusted to $b \geqslant 2k + 1$. Thus, without any loss of generality or precision, we can perform the analysis of the *k-error protocol* with respect to b, and then convert it to k, if needed.

In the analysis of the LZ code with the *k-error protocol*, Storer and Reif have conjectured that with constant b the LZ code should remain asymptotically optimal. That is, for any stationary ergodic source [3,18], the redundancy of this code will asymptotically tend to zero (with the length of code). While the asymptotic optimality of the code is an extremely important fact by itself, sometimes a stronger requirement is necessary: namely, that the (average) redundancy per symbol goes to zero at some universal rate. In this article we will derive a precise asymptotic expression for the expected redundancy rate of the LZ code with the *k-error protocol* for *memoryless sources*, which appears to be a rather complex function of both b and n (n is the length of an input sequence), attaining the asymptotic rate of $O\left(\log b / \log n\right)$. We will accompany our theoretical analysis with a number of numerical simulations that confirm the validity of our asymptotic expression, and show some interesting dependencies that could be useful for the design of practical algorithms.

2 Definitions and Main Results

In our analysis, we consider only an implementation of the *k-error protocol* with the original *Lempel-Ziv* scheme [19]. Storer and Reif [17] have demonstrated how to use the *k-error protocol* with various other modifications of the LZ algorithm [1], and we believe that it should be possible to extend our analysis to these cases as well.

Let α be the cardinality of an input alphabet, n be the length of an input sequence, M_n be the number of phrases in it, and M_n' be the number of distinct phrases constructed by the LZ scheme with the *k-error protocol*. Then the code length l_n produced by this scheme is:

$$l_n\left(b\right) = M_n\left(\log M_n' + \log \alpha\right) \tag{1}$$

The pointwise redundancy and its expected value of the LZ code with the *k-error protocol* are, respectively,

$$r_n(b) := \frac{l_n(b) - nh}{n} = \frac{M_n(\log M_n' + \log\alpha) - nh}{n}, \tag{2}$$

$$\bar{r}_n(b) := Er_n(b) = \frac{E[M_n(\log M_n' + \log\alpha)] - nh}{n}, \tag{3}$$

where h is the entropy rate of the source, and E is the expectation operator. Note that the base of the logarithms in the above formulas depend on the units of information being used. Thus, if we measure the code length l_n, entropy h, and redundancy r_n in nats, then all the logarithms should be natural; alternatively, if we use bits, the logarithms should be of base 2.

We now present the main theoretical result of this paper concerning the average redundancy rate $\bar{r}_n$ of the LZ code with the *k-error protocol* for memoryless sources.

Theorem 1 *Consider a memoryless source with alphabet $A = \{a_1, \ldots, a_\alpha\}$ and probabilities of symbols $P = \{p_1, \ldots, p_\alpha\}$; $\sum_{i=1}^{\alpha} p_i = 1$ (more specifically, if x_j is the j-th symbol produced by a source, then for any j: $\Pr\{x_j = a_i\} = p_i$). Then the average redundancy rate of the* LZ *code with the* k-error protocol *constructed using perfect hash function is:*

$$\bar{r}_n(b) = \frac{h}{\log n}\left(1 - \gamma - \frac{h_2}{2h} + \Delta(b,P) + H_{b-1} - \delta_1(n,b) + \right. \tag{4}$$
$$\left. \log(q_0(b) + \delta_2(n,b)) + \log\alpha\right) + O\left(\frac{\log\log n}{\log^2 n}\right),$$

where: b is a parsing redundancy constant that corresponds to the maximum number of errors k protected against propagation ($b = 3k$, for add- change- and remove- errors, or $b = 2k+1$, for change- errors only), $h = -\sum_{i=1}^{\alpha} p_i \log p_i$ is the entropy of the memoryless source, $h_2 = \sum_{i=1}^{\alpha} p_i \log^2 p_i$, $\gamma = 0.57721\ldots$ is the Euler constant, and $H_{b-1} = \sum_{i=1}^{b-1} i^{-1}$, $H_0 = 0$ are harmonic numbers. The constant $\Delta(b,P)$ can be computed as follows:

$$\Delta(b,P) = \sum_{n=2b+1}^{\infty} \bar{f}_n \sum_{i=1}^{b} \frac{(i+1)\,b!}{(b-1)!\,n(n-1)\ldots(n-i-1)} \tag{5}$$

where $\bar{f}_n$ is given recursively by:

$$\begin{cases} f_{m+b} = m + \sum_{i_1+\ldots+i_v=m} \binom{m}{i_1,\ldots,i_v} p_1^{i_1} \ldots p_v^{i_v} \left(f_{i_1} + \ldots + f_{i_v}\right), \ m > 0, \\ f_0 = f_1 = \ldots = f_b = 0, \\ \bar{f}_{m+b} = f_{m+b} - m, \ m \geqslant 1. \end{cases} \tag{6}$$

Another constant $q_0(b)$, in case of symmetric distribution ($p_i = 1/\alpha, i = 1 \ldots \alpha$), can be computed using the following expression [5]:

$$q_0(b) = \frac{1}{\ln 2} \int_0^{\infty} \left(\frac{1+t}{Q(t)}\right)^b \frac{dt}{1+t} \tag{7}$$

where $Q(t) = \prod_{j=0}^{\infty}\left(1 + t2^{-j}\right)$, and $\ln(.)$ is used to designate natural logarithm. In the asymmetric case, the constant $q_0(b)$ can be computed as suggested in [10]. Finally, $\delta_1(n,p)$ and $\delta_2(n,p)$ are fluctuating functions of small amplitude [11,5].

While formula (4) clearly demonstrates that the average redundancy rate $\bar{r}_n(b)$ decreases with the length of the input sequence n as $O(1/\log n)$, the dependence upon the constant b (and the corresponding number of errors, k) is much less obvious. Observe that when b is large, the largest term containing b in (4) is $H_{b-1} = \ln b + \gamma + O\left(\frac{1}{b}\right)$. The contribution of two other terms $\Delta(b,P)$ and $q_0(b)$ is much less significant. For example, in the symmetric case, it has been shown in [5] that for large b, $q_0(b) \sim 1/(b \ln 2)$, and in general $q_0(b) = o(1)$. The value of the constant $\Delta(b,P)$ is also bounded and decreases very slowly with b (cf. [13]). Combining all these terms, we conclude that for large b, the average redundancy rate of the *LZ* code with the *k-error protocol* is $O(\log b/\log n)$.

We will continue the discussion of the result of the Theorem 1 in Section 4, where we will also try to verify it experimentally.

3 Analysis

As it has been observed in [12], it is possible to model the *Lempel-Ziv* parsing scheme in two ways. The first is called *digital search tree* (or *DST*) *model*, and assumes that the number of phrases is given. For example, there are m statistically independent phrases, and one asks for a *LZ* sequence that can be built from these m phrases. Such construction can be done efficiently using a data structure called a *digital search tree*, as described in [9]. In another

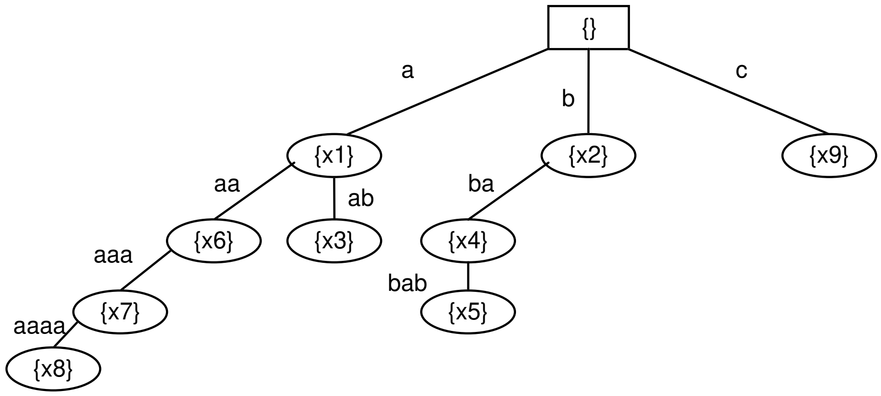


Fig. 1. A *digital search tree* built from strings $x_1 \ldots x_9$.

model, called the *Lempel-Ziv* (or *LZ*) *model*, there is only one sequence of fixed length n, and one partitions the sequence according to the *LZ* algorithm described above. Hereafter, we shall consistently use n as the length of a single string to be parsed, and m as the number of words (phrases) used to construct the *LZ* sequence.

Let us have a closer look at the *DST model*. Consider a case in which $m = 9$ sequences are: $x_1 = ababb\ldots$, $x_2 = babba\ldots$, $x_3 = abbab\ldots$, $x_4 = babab\ldots$, $x_5 = babaa\ldots$, $x_6 = aaaaaaaaac\ldots$, $x_7 = aaaaaaac\ldots$, $x_8 = aaaac\ldots$, $x_9 = caaaa\ldots$. We start the construction of a digital search tree with an empty root node. Then the insertion of a string x_1 will result in a new node corresponding to the branch $\{a\}$ from the root. This new node can now be used to insert any subsequent string that starts with symbol a (in our example it will be x_3), and so on. The resulting *DST* structure build from the strings $x_1 \ldots x_9$ is shown in Fig.1, and, as it is easy to observe, the corresponding *LZ* sequence is: *(a)(b)(ab)(ba)(bab)(aa)(aaa)(aaaa)(c)*.

Note that the digital search tree maintains only one copy of each string in its nodes, which corresponds precisely to the incremental parsing procedure of the original *LZ* scheme (cf. [19]). In the *LZ* scheme with the *k-error protocol*, however, the parsing is *redundant* (i.e. each phrase can be used several times), and thus, we will need an extension of the *DST* structure that can store multiple strings in its nodes. One such extension is known as *b-digital search tree*, (or *b-DST*) (cf. [5,10]). A *b-DST* structure starts with a root containing b empty strings, and each new node is created with b slots available for storing strings. During the construction, these slots are used to accumulate the first b strings that have the same prefix. Once all b slots are filled, a node can be used to create a new child node, and so on. An example of a *b-DST* built from the strings $x_1 \ldots x_9$ is shown in Fig.2. The corresponding *LZ* sequence is *(a)(b)(a)(b)(b)(a)(aa)(aa)(c)*.

Now we need to investigate if (and/or when) a *b-DST* is sufficient to model

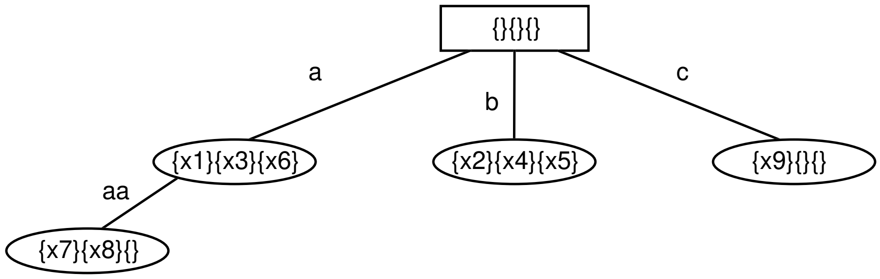


Fig. 2. A *b-DST* ($b = 3$) built from strings $x_1 \ldots x_9$.

the parsing scheme of the *LZ* code with the *k-error protocol*. Recall that by definition, the *k-error protocol* hashes all the phrases into a single table (dictionary), and that it uses a location in this table only when there is a clear winner among all the phrases hashed to it. More specifically, a winning phrase shall be encountered at least b times more than its nearest candidate. Observe that if hashing is *perfect*, only one phrase can be mapped into each location, and thus, the *k-error protocol* should just ensure that each phrase has been encountered b times. This is exactly the logic used in the *b-DST model*. In other words, we can conclude that a *b-DST* is sufficient to model the parsing scheme of the *LZ* code with the *k-error protocol* when hashing is perfect. Consequently, when hashing is not perfect, the *b-DST model* can still be used to study the lower bound of the redundancy rate.

Remarkably enough, a corresponding to the *b-DST model* extension of the *LZ model* has been recently studied in [13], however in a completely different context than the *k-error protocol*. Such an extension is called the *generalized Lempel-Ziv* (or *b-LZ*) *model*, and it has been proposed to study the *generalized LZ algorithm* introduced in [11]. Unlike the *k-error protocol* (which uses redundant parsing to prevent error propagation) the *generalized LZ algorithm* attempts to improve compression rate by using more compact encoding. Thus, when the *generalized LZ algorithm* creates a new phrase, it encodes it using the standard *(prefix pointer, symbol)* pair. However, the following $b - 1$ times this phrase is found, the *generalized LZ algorithm* encodes it as *(pointer)* only, saving $\log \alpha$ bits per each (reused) phrase.

In essence, having a b-redundant parsing of a source string, we can construct both the *generalized LZ* code and the *LZ* code with the *k-error protocol*, and thus, the use of the *b-DST* and *b-LZ* models should be adequate to study both algorithms. Below, we will enumerate some of the results obtained for *b-DST* and *b-LZ* models, and will derive an expression for the expected redundancy rate of the *LZ* code with the *k-error protocol* for memoryless sources, claimed by the Theorem 1. A similar result for the expected redundancy rate of the *generalized LZ* code has been obtained in [13].

Let $D_m(i)$ be the length of the ith phrase in the *b-DST model*. By D_m we denote the length of a randomly selected phrase, that is: $\Pr\{D_m = k\} = \frac{1}{m}\sum_{i=1}^{m}\Pr\{D_m(i) = k\}$. The length of a sequence built in the *b-DST model* is denoted by L_m, and it is equal to $L_m = \sum_{i=1}^{m} D_m(i)$. In the *b-LZ model*, we write D_n^{LZ} for the length of a randomly selected phrase, and M_n for the number of phrases constructed from a string of length n.

Some of the statistics of *digital search trees* and their *b-extensions* have been thoroughly analyzed (cf. [4,5,9–11]). Thus, if a *b-DST* is constructed from m independent strings generated by a biased *memoryless source*, it has been shown in [13] that the length of a randomly selected phrase has:

$$\begin{aligned} ED_m &= \frac{1}{h}\left(\log m + \frac{h_2}{2h} + \gamma - 1 - H_{b-1} - \Delta(b,P) + \delta_1(m,b)\right) \\ &\quad + O\left(\frac{\log m}{m}\right), \\ \operatorname{var} D_m &= c_2 \log m + O(1), \end{aligned} \tag{8}$$

where $c_2 = (h_2 - h^2)/h^3$, while h, h_2, H_{b-1}, $\Delta(b,P)$, and $\delta_1(m,b)$ are as defined in (3) and (4) of the Theorem 1. Similarly, the total *b-DST* path length L_m is known (cf. [6]) to have $EL_m = mED_m$, and $\operatorname{var} L_m = c_2 m \log m + O(m)$. Even more importantly, it has been found in [13] that for large m, D_m converges to a normal distribution:

$$\Pr\left\{D_m \leqslant ED_m + x\sqrt{\operatorname{var} D_m}\right\} = \Phi(x)\left(1 + O\left(1\Big/\sqrt{\log m}\right)\right) \tag{9}$$

In addition, an extension of the analysis of Jacquet and Szpankowski [6] reveals that:

$$\Pr\left\{L_m \leqslant EL_m + x\sqrt{\operatorname{var} L_m}\right\} = \Phi(x)\left(1 + O\left(1\Big/\sqrt{m}\right)\right) \tag{10}$$

for $x = O(1)$ (in the above formulas we denote $\Phi(x) = \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{x} e^{-t^2/2}dt$ for the distribution function of the standard normal distribution).

The situation is similar, but not the same, in the *LZ model*. Thus, to estimate the typical depth D_n^{LZ}, one will need to derive the limiting distribution of M_n. The last appears to be a very difficult problem (especially for $b > 1$), and authors of [6,12] have found a way to map this problem into another one, formulated in terms of *b-DST model*. The key in this transition is the following relationship between parameters of both models:

$$M_n = \max\left\{m : \; L_m = \sum_{i=1}^{m} D_m(i) \leqslant n\right\} \tag{11}$$

which immediately implies that :

$$\Pr\left[M_n > m\right] = \Pr\left[L_m \leqslant n\right] \tag{12}$$

The above relationship is known as the renewal equation and it has been extensively studied by Billingsley (cf. [2]).

Now, using (12) one can easily derive M_n, as well as all its moments (we will use M_n^k to denote k-th moment of M_n). Thus, as in [12]

$$\begin{aligned} EM_n^{k+1} &= (k+1)\sum_{m\geqslant 0} m^k \Pr\left\{M_n > m\right\} = (k+1)\sum_{m\geqslant 0} m^k \Pr\left\{L_m \leqslant n\right\} \\ &= (k+1)\int_0^\infty x^k \Pr\left\{L_x \leqslant n\right\} dx + O\left(EM_n\right) \end{aligned} \tag{13}$$

where the last follows from the Euler-Maclaurin formula (cf. [8]). By substituting $\Pr\left\{L_x \leqslant n\right\}$ with (10), and after long and tedious algebra, the authors of [12] have eventually arrived at:

$$EM_n = x_n\left(1 + O\left(\sqrt{\log n/n}\right)\right) + O\left(1\right) \tag{14}$$

where x_n is the solution of $EL_{x_n} = n$, namely

$$x_n = \frac{nh}{\log n}\left(1 + \frac{\log\log n}{\log n} + \frac{A - \log h}{\log n} + O\left(\frac{\left(\log\log n\right)^2}{\log^2 n}\right)\right) \tag{15}$$

where

$$A = 1 + H_{b-1} + \Delta\left(b, P\right) - \frac{h_2}{2h} - \gamma - \delta_1\left(m, b\right) \tag{16}$$

is h times the $\Theta\left(1\right)$ term in (8).

Finally, having listed all the above results, we can now derive the claim of the Theorem 1. According to (3), we will be mainly interested in two quantities: EM_n, and $E\left\{M_n \log M_n'\right\}$. The result for EM_n is listed in (14), while the second quantity can be further deduced to $E\left\{M_n \log M_n'\right\} = EM_n \log EM_n' + O\left(1/\log n\right)$ (cf. [12,13]). Here we will still need to estimate EM_n'. Observe that EM_n' is related to the size of the associated b-DST. More precisely, if S_n is the size of a b-DST with m strings, then $M_n' = S_{M_n} - 1$ (since we count only non-root nodes). However, according to Flajolet and Richmond [5] in the symmetric case, and Louchard [10] in the asymmetric case:

$ES_m = m\left(q_0\left(b\right) + \delta_2\left(m, b\right)\right) + O\left(1\right)$, where $q_0\left(b\right)$ is a constant that can be computed directly. Thus, asymptotically $EM'_n \sim x_n\left(q_0\left(b\right) + \delta_2\left(n, b\right)\right)$, and putting everything together we finally arrive at the formula (4) of the Theorem 1.

4 Experimental Results

In this section we will try to complement our analysis with a few experiments, aiming to verify the validity of our formula for the expected redundancy rate (4), and study effects of several additional modifications to the *LZ* scheme with the *k-error protocol*.

In our test model of the *LZ* algorithm, we have implemented *b-redundant parsing* and added various encoding schemes that can improve the performance of the algorithm for short sequences. Thus, our implementation supported the following modes:

(a) *k-error protocol*: sets $b = 2k + 1$ and encodes first b occurrences of a phrase as (prefix pointer, last symbol) pairs; uses direct pointer to this string afterwards;

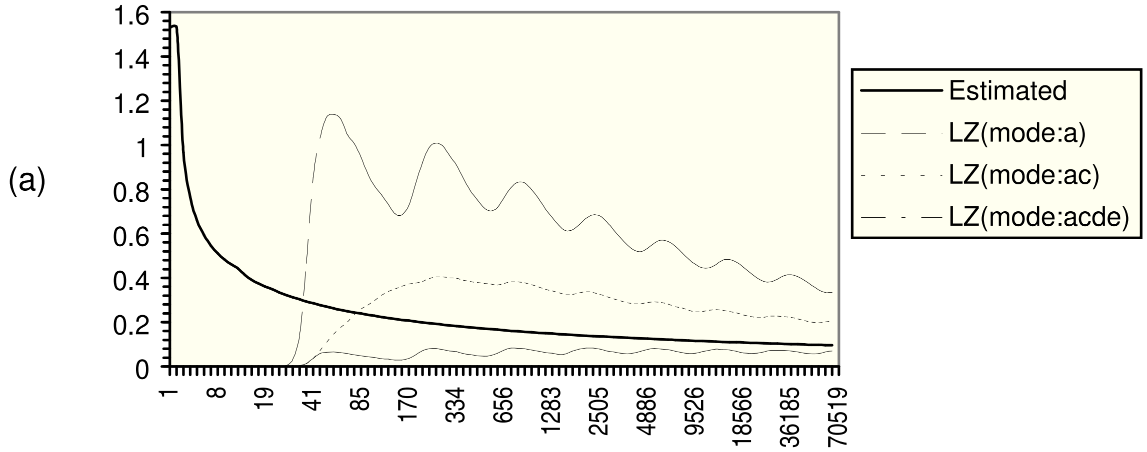
(b) *generalized LZ* (cf. [11]): when a new phrase is created it is encoded as *(prefix pointer, symbol)* pair, and a new pointer is assigned to it; the following $b - 1$ times this phrase is found it is encoded as *(pointer)* only, and no new phrase is created; after this period is over, it becomes a 'normal' phrase (i.e. every subsequent phrase will be encoded as *(pointer, symbol)* and added in the dictionary);

(c) adaptive encoding of pointers: if i is an index of a phrase, and i' is a current size of a dictionary, then i will be stored using $\lceil \log_2 i' \rceil$ bits;

(d) removal of internal nodes from the "encodable" part of the dictionary (cf. [15]): reduces size of the dictionary by eliminating phrases that are extended by all symbols in the alphabet (note: when the *k-error protocol* is used, the phrase can be eliminated only when all its children have been used at least b times);

(e) improved encoding of phrases with almost complete prefixes: if a prefix has already been extended by $\alpha - 1$ different symbols (α is the cardinality of the input alphabet), then it can only be used with the remaining symbol, and thus, in such a situation we can encode *(pointer)* only (with the *k-error protocol* we will have to encode *(pointer, symbol)* pair the first $b - 1$ times such phrase is used, and encode only a *(pointer)* when it is encountered the b-th time).

The charts of the pointwise redundancy rates for $b = 21$ and two types of source distribution (*symmetric* and *asymmetric* cases) are shown in Fig.3.(a)



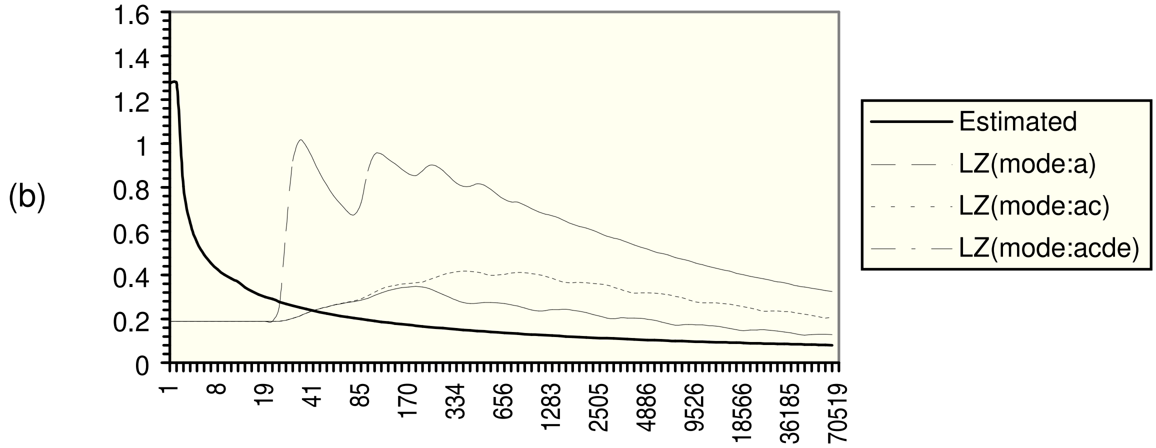


Fig. 3. Estimated vs. actual redundancy rates for various modifications of the *Lempel-Ziv code with the k-error protocol* for: $k = 10(b = 21)$. Input: binary memoryless source with: (a) $p = 0.5, h = 1$; (b) $p = 0.25, h = 0.81$.

and Fig.3.(b) correspondingly. Observe that when n is large, the differences between various modifications of the LZ code tend to become negligible and their redundancy rates tend to converge to a curve provided by our asymptotic expression (4). However, when n is small, the redundancy rates of actual algorithms are very different from one we have projected. Moreover, this is the situation when all the modifications we have listed above actually do make sense. Thus, we can notice that mode *(c)* can substantially reduce the redundancy at the beginning of a sequence. Two other modifications: mode *(d)* and mode *(e)* tent to provide a uniform reduction of redundancy across all n, however, such reduction depends on symmetry of the source. For instance, we were able to save almost 0.4 bits/symbol in a symmetric case (cf. Fig.3.a), while in a case with $p = 0.25$ (cf. Fig.3.b) this improvement was less than 0.2 bits/symbol.

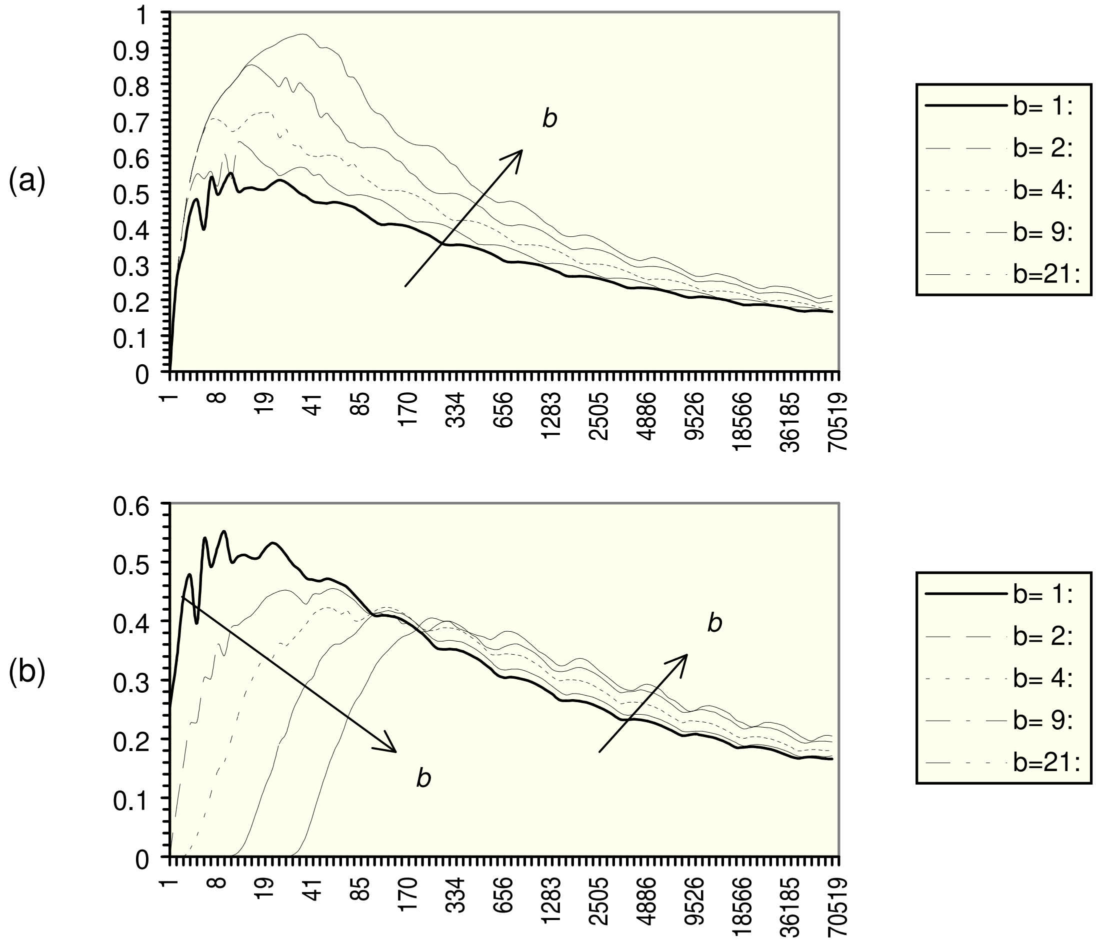


Fig. 4. Pointwise redundancy rates for various values of parameter b: (a) *generalized Lempel-Ziv code* vs. (b) *Lempel-Ziv with the k-error protocol*; Input: binary symmetric memoryless source ($p = 0.5$)

In Fig.4, we present a comparative analysis of two *b-redundant* codes: the *LZ* code with the *k-error protocol*, and the *generalized LZ code* (cf. [11]). In both cases we have used adaptive encoding of pointers (mode *(c)*. Based on our expression (4) we can expect that the redundancy rate of the *LZ* code with the *k-error protocol* to grow with b as $O(\log b / \log n)$. In addition, according to [11], we can expect the *generalized LZ code* to be less redundant than the *k-error protocol*. However, as we will see, both of these conjectures do not hold for the short sequences. Thus, according to Fig.4.b, for very small n, the redundancy of the *k-error protocol* does in fact, decrease with b. Then at some point (in our example, around $n = 200$) the redundancy rate becomes almost independent of b, and finally, for large n, the redundancy rate tends to grow consistently with b. On the contrary, the redundancy rate of the *generalized LZ* tends to grow very intensively with b around small n. In part, this anomaly can be explained by the difference in the way both algorithms add new phrases

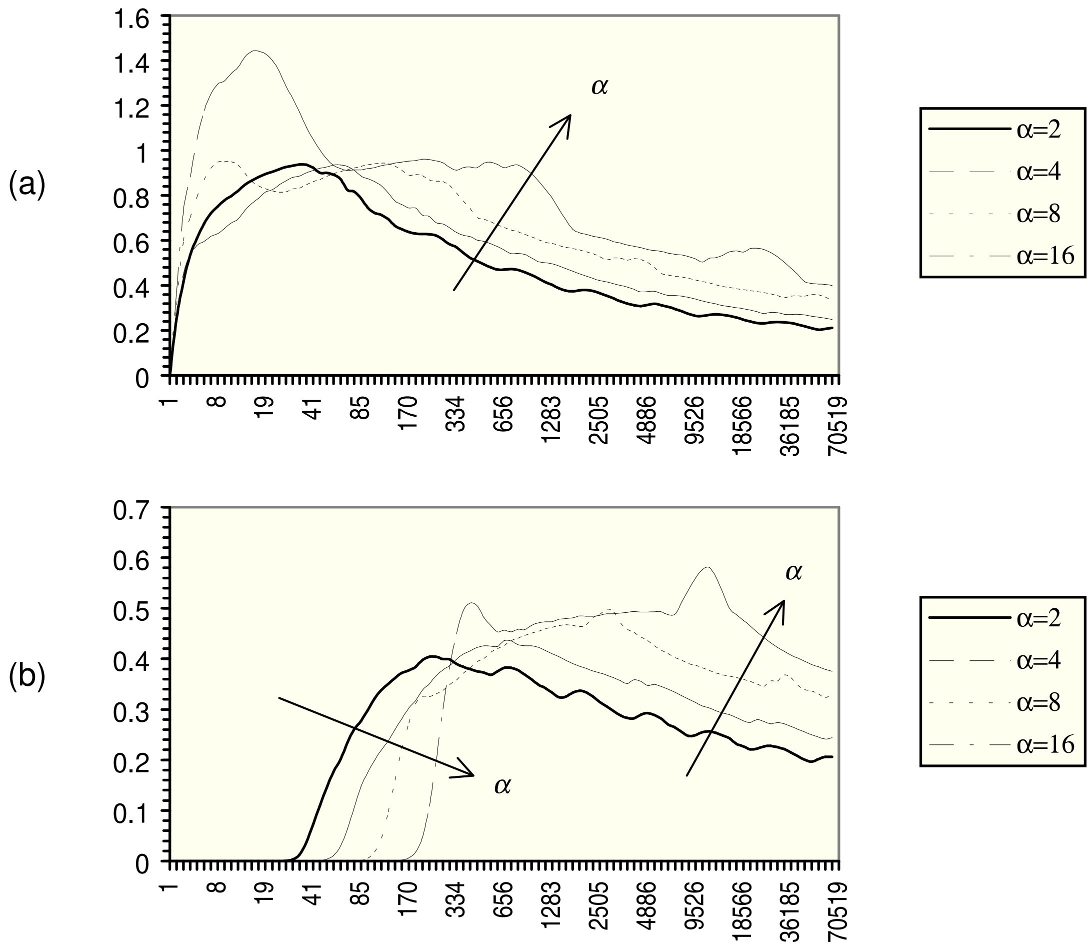


Fig. 5. Pointwise redundancy rates for symmetric memoryless sources over larger alphabets ($\alpha = 2, \ldots, 16$): (a) *generalized Lempel-Ziv code* vs. (b) *Lempel-Ziv with the k-error protocol*; In both cases the parsing redundancy is $b = 21$.

in the dictionary. Thus, since LZ with the *k-error protocol* does not encode a new phrase until it has been encountered first b times its addition to the dictionary is postponed. The *generalized* LZ, however, adds every new phrase immediately, which increases the size of the dictionary. This effect is most noticeable when pointers are encoded adaptively.

Finally, in Fig.5, we illustrate the behaviour of the LZ code with the *k-error protocol*, and the *generalized* LZ *code* with symmetric memoryless sources built over larger alphabets ($\alpha = 2 \ldots 16$). When n is large, the redundancy rates of both codes tend to grow consistently with α. Such behaviour can be fully predicted from our expression (4) (in case of the code with the *k-error protocol*) and results provided in [13] (for *generalized* LZ). However, the behaviour of these codes on short sequences (n is small) is quite different. Thus, the redundancy rate of the LZ code with the *k-error protocol* actually decreases with α for very small n (in our example it happens for $n \leqslant 200$). The situation with *generalized LZ* is even more complicated (cf. Fig.5.a). For example, when $n \leqslant 60$, the redundancy rate of the 4-ary *generalized LZ* code, seems to be lower than one for binary code. At the same time the redundancy of the 8-ary code can be either greater or lower than both 4-ary and binary codes, depending on the subrange of n in the same interval ($n \leqslant 60$). This property can be utilized in practice for selecting the optimal parsing redundancy constant (b) when the cardinality of the alphabet and the size of the input are given.

5 Extensions and Concluding Remarks

While the analysis that we have provided in this paper has many obvious limitations, we believe that it can be used as a tool for studying the behavior of many practical implementations of the *k-error protocol*. In part, we tried to demonstrate this in Section 4, where we evaluated several additional enhancements to this algorithm.

The extension of our theoretical result to *Markovian* sources can be a complex but feasible task. In fact, such an extension for a case of $b = 1$ (the original *Lempel-Ziv* scheme) has already been obtained in [12], and, using a different theoretical approach, in [14]. Several additional results in this direction have been recently reported in [7].

References

[1] T. Bell, J. Cleary, and I. Witten, Text Compression (Prentice Hall, Englewood Cliffs, NJ, 1990).

[2] P. Billingsley, Convergence of Probability Measures (Wiley, New York, 1991).

[3] T. M. Cover and J. M. Thomas, Elements of Information Theory (John Wiley & Sons, New York, 1991).

[4] P. Flajolet and R. Sedgewick, Digital Search Trees Revisited, SIAM J. Computing, 15 (1986) 748-767.

[5] P. Flajolet and B. Richmond, Generalized Digital Trees and Their Difference-Differential Equations, Random Structures and Algorithms, 3 (1992) 305-320.

[6] P. Jacquet and W. Szpankowski, Asymptotic Behavior of the Lempel-Ziv Parsing Scheme and Digital Search Trees, Theoretical Computer Science, 144 (1995) 161-197.

[7] P. Jacquet, W. Szpankowski, and J. Tang, Average Profile of the Lempel-Ziv Parsing Scheme for a Markovian Source, *Algorithmica*, to appear.

[8] D. Knuth, The Art of Computer Programming. Fundamental Algorithms. Vol. 1 (Addison-Wesley, 1968).

[9] D. Knuth, The Art of Computer Programming. Sorting and Searching. Vol. 3 (Addison-Wesley, 1973).

[10] G. Louchard, Digital Search Trees Revisited, Cahiers du CERO, 36 (1995) 259-27.

[11] G. Louchard and W. Szpankowski, Generalized Lempel-Ziv Parsing Scheme and its Preliminary Analysis of the Average Profile, Proc. IEEE Data Compression Conference, Snowbird, UT (1995) 262-271.

[12] G. Louchard and W. Szpankowski, On the Average Redundancy Rate of the Lempel-Ziv Code, IEEE Trans. Information Theory, 43 (1997) 2-8.

[13] G. Louchard, W. Szpankowski, and J. Tang, Average Profile of the Generalized Digital Search Tree and the Generalized Lempel-Ziv Algorithm, SIAM J. Computing, 28 (1999) 935-954.

[14] S. Savari, Redundancy of the Lempel-Ziv Incremental Parsing Rule, IEEE Trans. Information Theory, 43, (1997) 9-21.

[15] J. B. Seery and J. Ziv, Further Results on Universal Data Compression, Tech. Memorandum 78-1212-8 (Bell Laboratories, Murray Hill, NJ, 1978).

[16] J. Storer, Data Compression: Methods and Theory, (Computer Science Press, Rockville, MD, 1988).

[17] J. Storer and J. Reif, Error Resilient Optimal Data Compression, SIAM J. Computing, 26, 4 (1997) 934-939.

[18] A. Wyner and J. Ziv, Some Asymptotic Properties of the Entropy of a Stationary Ergodic Data Source with Application to Data Compression, IEEE Trans. Information Theory, 35 (1989) 1250-1258.

[19] J. Ziv and A. Lempel, Compression of Individual Sequences via Variable-Rate Coding, IEEE Trans. Information Theory, 24 (1978) 550-536.