Algorithmica manuscript No.
(will be inserted by the editor)

Lempel–Ziv-like Parsing in Small Space

**Dmitry Kosolobov · Daniel Valenzuela ·
Gonzalo Navarro · Simon J. Puglisi**

Received: date / Accepted: date

Abstract Lempel–Ziv (LZ77 or, briefly, LZ) is one of the most effective and widely-used compressors for repetitive texts. However, the existing efficient methods computing the exact LZ parsing have to use linear or close to linear space to index the input text during the construction of the parsing, which is prohibitive for long inputs. An alternative is Relative Lempel–Ziv (RLZ), which indexes only a fixed reference sequence, whose size can be controlled. Deriving the reference sequence by sampling the text yields reasonable compression ratios for RLZ, but performance is not always competitive with that of LZ and depends heavily on the similarity of the reference to the text. In this paper we introduce ReLZ, a technique that uses RLZ as a preprocessor to approximate the LZ parsing using little memory. RLZ is first used to produce a sequence of phrases, and these are regarded as metasymbols that are input to LZ for a second-level parsing on a (most often) drastically shorter sequence. This parsing is finally translated into one on the original sequence.

D. Kosolobov supported by the Russian Science Foundation (RSF), project 18-71-00002 (for the upper bound analysis and a part of lower bound analysis). D. Valenzuela supported by the Academy of Finland (grant 309048). G. Navarro funded by Basal Funds FB0001 and Fondecyt Grant 1-200038, Chile. S.J. Puglisi supported by the Academy of Finland (grant 319454).

D. Kosolobov*
Ural Federal University, Ekaterinburg, Russia
E-mail: dkosolobov@mail.ru

D. Valenzuela
Department of Computer Science, University of Helsinki, Finland
E-mail: dvalenzu@cs.helsinki.fi

G. Navarro
CeBiB, Department of Computer Science, University of Chile, Chile
E-mail: gnavarro@dcc.uchile.cl

S.J. Puglisi
Department of Computer Science, University of Helsinki, Finland
E-mail: puglisi@cs.helsinki.fi

We analyze the new scheme and prove that, like LZ, it achieves the kth order empirical entropy compression $nH_k+o(n\log\sigma)$ with $k=o(\log_\sigma n)$, where n is the input length and σ is the alphabet size. In fact, we prove this entropy bound not only for ReLZ but for a wide class of LZ-like encodings. Then, we establish a lower bound on ReLZ approximation ratio showing that the number of phrases in it can be $\Omega(\log n)$ times larger than the number of phrases in LZ. Our experiments show that ReLZ is faster than existing alternatives to compute the (exact or approximate) LZ parsing, at the reasonable price of an approximation factor below 2.0 in all tested scenarios, and sometimes below 1.05, to the size of LZ.

Keywords Lempel–Ziv compression · Relative Lempel–Ziv · empirical entropy

1 Introduction

The Lempel–Ziv (LZ77 or, shortly, LZ) parsing is a central algorithm in data compression: more than 40 years since its development [49,50], it is at the core of widely used compressors (gzip, p7zip, zip, arj, rar...) and compressed formats (PNG, JPEG, TIFF, PDF...), and receives much attention from researchers [3, 19,21,39,40] and developers in industry [1,47].

LZ parsing also has important theoretical properties. The number of phrases, z, into which LZ parses a text has become the defacto measure of compressibility for dictionary-based methods [14], which in particular are most effective on highly repetitive sequences [36]. While there are measures that are stronger than LZ [42,23], these are NP-complete to compute. The LZ parse, which can be computed greedily in linear time [20], is then the stronger measure of dictionary-based compressibility on which to build practical compressors.

Computing the LZ parsing requires the ability to find previous occurrences of text substrings (their "source"), so that the compressor can replace the current occurrence (the "target") by a backward pointer to the source. Parsing in linear time [20] requires building data structures that are proportional to the text size. When the text size exceeds the available RAM, switching to external memory leads to prohibitive computation times. Compression utilities avoid this problem with different workarounds: by limiting the sources to lie inside a short sliding window behind the current text (see [13,41,46]) or by partitioning the input into blocks and compressing them independently. These variants can greatly degrade compression performance, however, and are unable in particular to exploit long-range repetitions.

Computation of LZ in compressed space was first studied—to the best of our knowledge—in 2015: A $(1+\epsilon)$-approximation scheme running in $O(n\log n)$ time[1] with $O(z)$ memory, where n is the length of the input, was proposed in [11], and an exact algorithm with the same time $O(n\log n)$ and space bounded by the zeroth order empirical entropy was given in [38]. The work [22]

[1] Hereafter, log denote logarithm with base 2 if it is not explicitly stated otherwise.

shows how to compute LZ-End—an LZ-like parsing—using $O(z + \ell)$ compressed space and $O(n \log \ell)$ time w.h.p., where ℓ is the length of the longest phrase. The recent studies on the Run-Length Burrows–Wheeler Transform (RLBWT) [15] and its connections to LZ [3,39] have enabled the computation of the LZ parsing in compressed space $O(r)$ and time $O(n \log r)$ via RLBWT, where r is the number of runs in the RLBWT.

Relative Lempel–Ziv (RLZ) [27] is a variant of LZ that exploits another approach: it uses a fixed external sequence, the *reference*, where the sources are to be found, which performs well when the reference is carefully chosen [16,33]. Different compressors have been proposed based on this idea [6,7,33]. When random sampling of the text is used to build an artificial reference the expected encoded size of the RLZ relates to the size of LZ [16], however the gap is still large in practice. Some approaches have done a second pass of compression after RLZ [19,6,45] but they do not produce an LZ-like parsing that could be compared with LZ.

In this paper we propose ReLZ, a parsing scheme that approximates the LZ parsing by making use of RLZ as a preprocessing step. The phrases found by RLZ are treated as metasymbols that form a new sequence, which is then parsed by LZ to discover longer-range repetitions. The final result is then expressed as phrases of the original text. The new sequence on which LZ is applied is expected to be much shorter than the original, which avoids the memory problems of LZ. In exchange, the parsing we obtain is limited to choose sources and targets formed by whole substrings found by RLZ, and is therefore suboptimal.

We analyze the new scheme and prove that, like LZ, it achieves the kth order empirical entropy compression $nH_k + o(n \log \sigma)$ (see definitions below) with $k = o(\log_\sigma n)$, where n is the length of the input string and σ is the alphabet size. We show that it is crucial for this result to use the so-called rightmost LZ encoding [2,4,5,10,29] in the second step of ReLZ; to our knowledge, this is the first provable evidence of the impact of the rightmost encoding. In fact, the result is more general: we show that the rightmost encoding of any LZ-like parsing with $O(\frac{n}{\log_\sigma n})$ phrases achieves the entropy compression when a variable length encoder is used for phrases. One might interpret this as an indication of the weakness of the entropy measure. We then relate ReLZ to LZ—the de facto standard for dictionary-based compression—and prove that the number of phrases in ReLZ might be $\Omega(z \log n)$; we conjecture that this lower bound is tight. The new scheme is tested and, in all the experiments, the number of phrases found by ReLZ never exceeded $2z$ (and it was around $1.05z$ in some cases). In exchange, ReLZ computes the parsing faster than the existing alternatives.

The paper is organized as follows. In Sections 2 and 3 we introduce some notation and define the ReLZ parsing and its variations. Section 4 contains the empirical entropy analysis. Section 5 establishes the $\Omega(z \log n)$ lower bound. All experimental results are in Sections 6 and 7.

2 Preliminaries

Let $T[1, n]$ be a string of length n over the alphabet $\Sigma = \{1, 2, \ldots, \sigma\}$; $T[i]$ denotes the ith symbol of T and $T[i, j]$ denotes the substring $T[i]T[i{+}1] \cdots T[j]$. A substring $T[i, j]$ is *a prefix* if $i = 1$ and *a suffix* if $j = n$. The *reverse* of T is the string $T[n]T[n-1] \cdots T[1]$. The concatenation of two strings T and T' is denoted by $T \cdot T'$ or simply TT'.

The *zeroth order empirical entropy* (see [24,35]) of $T[1, n]$ is defined as $H_0(T) = \sum_{c \in \Sigma} \frac{n_c}{n} \log \frac{n}{n_c}$, where n_c is the number of symbols c in T and $\frac{n_c}{n} \log \frac{n}{n_c} = 0$ whenever $n_c = 0$. For a string W, let T_W be a string formed by concatenating all symbols immediately following occurrences of W in $T[1, n]$; e.g., $T_{ab} = aac$ for $T = abababc$. The *kth order empirical entropy* of $T[1, n]$ is defined as $H_k(T) = \sum_{W \in \Sigma^k} \frac{|T_W|}{n} H_0(T_W)$, where Σ^k is the set of all strings of length k over Σ (see [24,34,35]). If T is clear from the context, $H_k(T)$ is denoted by H_k. It is well known that $\log \sigma \geq H_0 \geq H_1 \geq \cdots$ and H_k makes sense as a measure of string compression only for $k < \log_\sigma n$ (see [12] for a deep discussion).

The *LZ parsing* [49] of $T[1, n]$ is a sequence of non-empty *phrases* (substrings) $LZ(T) = (P_1, P_2, \ldots, P_z)$ such that $T = P_1 P_2 \cdots P_z$, built as follows. Assuming we have already parsed $T[1, i-1]$, producing $P_1, P_2, \ldots, P_{j-1}$, then P_j is set to the longest prefix of $T[i, n]$ that has a previous occurrence in T that starts before position i. Such a phrase P_j is called a *copying phrase*, and its previous occurrence in T is called *the source* of P_j. When the longest prefix is of length zero, the next phrase is the single symbol $P_j = T[i]$, and P_j is called a *literal phrase*. This greedy parsing strategy yields the least number of phrases (see [49, Th. 1]).

LZ compression consists in replacing copying phrases by backward pointers to their sources in T, and T can obviously be reconstructed in linear time from these pointers. A natural way to encode the phrases is as pairs of integers: for copying phrases P_j, a pair (d_j, ℓ_j) gives the distance to the source and its length, i.e., $\ell_j = |P_j|$ and $T[|P_1 \cdots P_{j-1}| - d_j + 1, n]$ is prefixed by P_j; for literal phrases $P_j = c$, a pair $(c, 0)$ encodes the symbol c as an integer. Such encoding is called *rightmost* if the numbers d_j in all the pairs (d_j, ℓ_j) are minimized, i.e., the rightmost sources are chosen.

When measuring the compression efficiency of encodings, it is natural to assume that σ is a non-decreasing function of n. In such premises, if each d_j component occupies $\lceil \log n \rceil$ bits and each ℓ_j component takes $O(1 + \log \ell_j)$ bits, then it is known that the size of the LZ encoding is upperbounded by $nH_k + o(n \log \sigma)$ bits, provided k is a function of n such that $k = o(\log_\sigma n)$; see [17,24,37]. In the sequel we also utilize a slightly different encoding that, for each d_j, uses a universal code [9,32] taking $\log d_j + O(1 + \log \log d_j)$ bits.

Other parsing strategies that do not necessarily choose the longest prefix of $T[i, n]$ are valid, in the sense that T can be recovered from the backward pointers. Those are called *LZ-like parses*. Some examples are LZ-End [26], which forces sources to finish at the end of a previous phrase, LZ77 with sliding window [50], which restricts the sources to start in $T[i - w, i - 1]$ for a fixed windows size w, and the bit-optimal LZ [10, 25], where the phrases are chosen to minimize the encoding size for a given encoder of pairs.

The RLZ parsing [27] of $T[1,n]$ with reference $R[1,\ell]$ is a sequence of phrases $RLZ(T,R) = (P_1, P_2, \ldots, P_z)$ such that $T = P_1 P_2 \cdots P_z$, built as follows: Assuming we have already parsed $T[1, i-1]$, producing $P_1, P_2, \ldots, P_{j-1}$, then P_j is set to the longest prefix of $T[i,n]$ that is a substring of $R[1,\ell]$; by analogy to the LZ parsing, P_j is a *copying phrase* unless it is of length zero; in the latter case we set $P_j = T[i]$, a *literal phrase*. Note that RLZ does not produce an LZ-like parsing as we have defined it.

3 ReLZ Parsing

First we present RLZ$_{pref}$ [44], a variant of RLZ that instead of using an external reference uses a prefix of the text as a reference to produce an LZ-like parsing. The RLZ$_{pref}$ parsing of T, given a parameter ℓ, is defined as $RLZ_{prefix}(T,\ell) = LZ(T[1,\ell]) \cdot RLZ(T[\ell+1,n], T[1,\ell])$. That is, we first compress $T[1,\ell]$ with LZ, and then use that prefix as the reference to compress the rest, $T[\ell+1,n]$, with RLZ. Note that RLZ$_{pref}$ is an LZ-like parsing.

The ReLZ algorithm works as follows. Given a text $T[1,n]$ and a prefix size ℓ, we first compute the RLZ$_{pref}$ parsing $(P_1, P_2, \ldots, P_{z'})$ (so that $T = P_1 P_2 \cdots P_{z'}$). Now we consider the phrases P_j as atomic metasymbols, and define a string $T'[1,z']$ such that, for every i and j, $T'[i] = T'[j]$ iff $P_i = P_j$. Then we compress $T'[1,z']$ using LZ, which yields a parsing $(P'_1, P'_2, \ldots, P'_{\hat{z}})$ of T'. Finally, the result is transformed into an LZ-like parsing of T in a straightforward way: each literal phrase P'_j corresponds to a single phrase P_i and, thus, is left unchanged; each copying phrase P'_j has a source $T'[p,q]$ and is transformed accordingly into a copying phrase in T with the source $T[p',q']$, where $p' = |P_1 P_2 \cdots P_{p-1}| + 1$ and $q' = |P_1 P_2 \cdots P_q|$. Figure 1 shows an example.

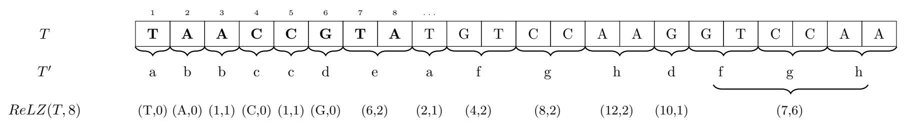

1 2 3 4 5 6 7 8 ...
T
T A A C C G T A T G T C C A A G G T C C A A
T'
a b b c c d e a f g h d f g h
$ReLZ(T,8)$
(T,0) (A,0) (1,1) (C,0) (1,1) (G,0) (6,2) (2,1) (4,2) (8,2) (12,2) (10,1) (7,6)

Fig. 1 An example of ReLZ, using prefix size $\ell = 8$. The first line below the text shows the string T' corresponding to the RLZ$_{pref}$ parsing. Note that the substring "GTCCAA" occurs twice, but RLZ$_{pref}$ misses this repetition because there is no similar substring in the reference. Nonetheless, both occurrences are parsed identically. The string T' is then parsed using LZ. The latter captures the repetition of the sequence "fgh", and when this parsing is remapped to the original text, it captures the repetition of "GTCCAA".

Since both LZ [20] and RLZ [28] run in linear time, ReLZ can also be implemented in time $O(n)$.

Obviously, the first ℓ symbols do not necessarily make a good reference for the RLZ step in ReLZ. In view of this, it seems practically relevant to define the following variant of ReLZ: given a parameter $\ell = o(n)$, we first sample in a certain way (for instance, randomly as in [16]) disjoint substrings of T with total length ℓ, then concatenate them making a string A of length ℓ, and apply ReLZ to the string AT; the output encoding of T is the $\lceil \log n \rceil$-bit number ℓ followed by an encoding of the LZ-like parsing of AT produced by ReLZ. Nevertheless, throughout the paper we concentrate only on the first version of ReLZ, which generates an LZ-like parsing. This choice is justified by two observations: first, it is straightforward that the key part in any analysis of the second ReLZ variant is in the analysis of ReLZ for the string AT; and second, our experiments on real data comparing known sampling methods (see Section 7.3) show that the first version of ReLZ leads to better compression, presumably because the improvements made by the sampling in the RLZ step do not compensate for the need to keep the reference A.

4 Empirical Entropy Upper Bound

Our entropy analysis relies on the following lemmas by Gagie [12], Ochoa and Navarro [37], and Gańczorz [17].

Lemma 1 ([12, Th. 1]) *For any string $T[1, n]$ and any integer $k \geq 0$, $nH_k(T) = \min\{\log(1/Pr(Q \text{ emits } T))\}$, where the minimum is over all kth order Markov processes Q.*

Lemma 2 ([17] and [37, Lm. 3]) *Let Q be a kth order Markov process. Any parsing $T = P_1 P_2 \cdots P_c$ of a given string $T[1, n]$ over the alphabet $\{1, 2, \ldots, \sigma\}$, where all P_i are non-empty, satisfies:*

$$\sum_{i=1}^{c} \log \frac{c}{c_i} \leq \log \frac{1}{Pr(Q \text{ emits } T)} + O(ck \log \sigma + c \log \frac{n}{c}),$$

where c_i is the number of times P_i occurs in the sequence $P_1, P_2, \ldots, P_c$.

Recall that in this discussion σ and k in H_k both are functions of n. Now we are to prove that, as it turns out, the kth order empirical entropy is easily achievable by any LZ-like parsing in which the number of phrases is $O(\frac{n}{\log_\sigma n})$: it suffices to use the rightmost encoding and to spend at most $\log d_j + O(1 + \log\log d_j + \log \ell_j)$ bits for every pair (d_j, ℓ_j) corresponding to a copying phrase (for instance, applying for d_j and ℓ_j universal codes, like Elias's [9] or Levenshtein's [32]). In the sequel we show that, contrary to the case of LZ (see [24,37]), it is not possible to weaken the assumptions in this result—even for ReLZ—neither by using a non-rightmost encoding nor by using $\log n + O(1 + \log \ell_j)$ bits for the pairs (d_j, ℓ_j).

Lemma 3 *Fix a constant $\alpha > 0$. Given a string $T[1, n]$ over the alphabet $\{1, 2, \dots, \sigma\}$ with $\sigma \le O(n)$ and its LZ-like parsing $T = P_1 P_2 \cdots P_c$ such that $c \le \frac{\alpha n}{\log_\sigma n}$, the rightmost encoding of the parsing in which every pair (d_j, ℓ_j) corresponding to a copying phrase takes $\log d_j + O(1 + \log\log d_j + \log \ell_j)$ bits occupies at most $nH_k + o(n \log \sigma)$ bits, for $k = o(\log_\sigma n)$.*

Proof First, let us assume that k is a positive function of n, $k > 0$. Since $k = o(\log_\sigma n)$, it implies that $\log_\sigma n = \omega(1)$ and $\sigma = o(n)$. Therefore, all literal phrases occupy $O(\sigma \log \sigma) = o(n \log \sigma)$ bits. For $i \in \{1, 2, \dots, c\}$, denote by c_i the number of times P_i occurs in the sequence $P_1, P_2, \dots, P_c$. Let $P_{i_1}, P_{i_2}, \dots, P_{i_{c_i}}$ be the subsequence of all phrases equal to P_i. Since the encoding we consider is rightmost, we have $d_{i_1} + d_{i_2} + \cdots + d_{i_{c_i}} \le n$. Therefore, by the concavity of the function log, we obtain $\log d_{i_1} + \log d_{i_2} + \cdots + \log d_{i_{c_i}} \le c_i \log \frac{n}{c_i} = c_i \log \frac{n}{c} + c_i \log \frac{c}{c_i}$. Similarly, we deduce $\log\log d_{i_1} + \log\log d_{i_2} + \cdots + \log\log d_{i_{c_i}} \le c_i \log\log \frac{n}{c_i}$ and $\sum_{j=1}^{c} \log \ell_j \le c \log \frac{n}{c}$. Hence, the whole encoding occupies $\sum_{i=1}^{c} (\log \frac{c}{c_i} + O(\log\log \frac{n}{c_i})) + O(c \log \frac{n}{c}) + o(n \log \sigma)$ bits. By Lemmas 1 and 2, this sum is upperbounded by

$$nH_k + O(c \log \frac{n}{c} + ck \log \sigma + \sum_{i=1}^{c} \log\log \frac{n}{c_i}) + o(n \log \sigma). \qquad (1)$$

It remains to prove that all the terms under the big-O are $o(n \log \sigma)$. Since $k = o(\log_\sigma n)$ and $c \le \frac{\alpha n}{\log_\sigma n}$, we have $ck \log \sigma \le o(n \log \sigma)$. As $c \log \frac{n}{c}$ is an increasing function of c when $c < n/2$, we obtain $c \log \frac{n}{c} \le O(\frac{n}{\log_\sigma n} \log \log_\sigma n) = o(n \log \sigma)$. Further, $\log\log \frac{n}{c_i} = \log(\log \frac{n}{c} + \log \frac{c}{c_i}) \le \log\log \frac{n}{c} + O(\log \frac{c}{c_i} / \log \frac{n}{c})$ due to the inequality $\log(x + d) \le \log x + \frac{d \log e}{x}$. The sum $\sum_{i=1}^{c} \log\log \frac{n}{c}$ is upperbounded by $c \log \frac{n}{c} = o(n \log \sigma)$. The sum $\sum_{i=1}^{c} \log \frac{c}{c_i} / \log \frac{n}{c}$ is upperbounded by $(c \log c) / \log \frac{n}{c}$, which can be estimated as $O((n \log \sigma) / \log \log_\sigma n)$ because $c \le \frac{\alpha n}{\log_\sigma n}$. Since $\log_\sigma n = \omega(1)$, this is again $o(n \log \sigma)$.

Now assume that $k = 0$; note that in this case $k = o(\log_\sigma n)$ even for $\sigma = \Omega(n)$. It is sufficient to consider only the case $\sigma > 2^{\sqrt{\log n}}$ since, for $\sigma \le 2^{\sqrt{\log n}}$, we have $\sigma \log \sigma = o(n \log \sigma)$ and $\log_\sigma n \ge \sqrt{\log n} = \omega(1)$ and, hence, the above analysis is applicable. As σ can be close to $\Theta(n)$, the literal phrases might now take non-negligible space. Let A be the subset of all symbols $\{1, 2, \dots, \sigma\}$ that occur in T. For $a \in A$, denote by i_a the leftmost phrase $P_{i_a} = a$. Denote $C = \{1, 2, \dots, c\} \setminus \{i_a \colon a \in A\}$, the indices of all copying phrases. The whole encoding occupies at most $\sum_{i \in C} (\log d_i + O(1 + \log\log d_i + \log \ell_i)) + |A| \log \sigma + O(|A|(1 + \log\log \sigma))$ bits, which is upperbounded by $\sum_{i \in C} \log d_i + |A| \log |A| + O(n + n \log\log n + c \log \frac{n}{c}) + |A| \log \frac{\sigma}{|A|}$. Observe that $n \log\log n \le o(n\sqrt{\log n}) \le o(n \log \sigma)$. Further, we have $c \log \frac{n}{c} \le n$ and $|A| \log \frac{\sigma}{|A|} \le \sigma$, both of which are $o(n \log \sigma)$ since $\sigma \le O(n) \le o(n \log \sigma)$. It remains to bound $\sum_{i \in C} \log d_i + |A| \log |A|$ with $nH_0 + o(n \log \sigma)$.

Let us show that $|A| \log |A| \le \sum_{a \in A} \log \frac{n}{c_{i_a}}$. Indeed, we have $\sum_{a \in A} \log \frac{n}{c_{i_a}} = \log(n^{|A|} / \prod_{a \in A} c_{i_a})$, which, since $\sum_{a \in A} c_{i_a} \le n$, is minimized when all c_{i_a} are equal to $\frac{n}{|A|}$ so that $\sum_{a \in A} \log \frac{n}{c_{i_a}} \ge |A| \log |A|$. For $i \in C$, denote by

c'_i the number of copying phrases equal to P_i, i.e., $c'_i = c_i$ if $|P_i| > 1$, and $c'_i = c_i - 1$ otherwise (note that $c'_i > 0$ for all $i \in C$). As in the analysis for $k > 0$, we obtain $\sum_{i\in C} \log d_i \le \sum_{i\in C} \log \frac{n}{c'_i}$. Fix $i \in C$ such that $|P_i| = 1$. Using the inequality $\log(x-d) \ge \log x - \frac{d \log e}{x-d}$, we deduce $\log \frac{n}{c'_i} = -\log(\frac{c_i}{n} - \frac{1}{n}) \le \log \frac{n}{c_i} + \frac{\log e}{c'_i}$. Therefore, $|A| \log |A| + \sum_{i\in C} \log \frac{n}{c'_i} \le \sum_{i=1}^{c} \log \frac{n}{c_i} + c \log e = \sum_{i=1}^{c} \log \frac{c}{c_i} + c \log \frac{n}{c} + O(n)$. By Lemmas 1 and 2, this is upperbounded by (1). As $k = 0$, the terms under the big-O of (1) degenerate to $c \log \frac{n}{c} + \sum_{i=1}^{c} \log\log \frac{n}{c_i}$, which is $O(n \log\log n) \le o(n \log \sigma)$. □

It follows from the proof of Lemma 3 that, instead of the strict rightmost encoding, it is enough to choose, for each copying phrase P_j of the LZ-like parsing, the closest preceding equal phrase—i.e., $P_i = P_j$ with maximal $i < j$—as a source of P_j, or any source if there is no such P_i. This observation greatly simplifies the construction of an encoding that achieves the H_k bound. Now let us return to the discussion of the ReLZ parsing.

Lemma 4 *The number of phrases in the ReLZ parsing of any string $T[1,n]$ over the alphabet $\{1,2,\ldots,\sigma\}$ is at most $\frac{9n}{\log_\sigma n}$, independent of the choice of the prefix parameter ℓ.*

Proof For $\sigma \ge n^{1/9}$, we have $\frac{9n}{\log_\sigma n} \ge n$ and, hence, the claim is obviously true. Assume that $\sigma < n^{1/9}$. As $\sigma \ge 2$, this implies $n > 2^9 = 512$. Suppose that $T = P_1 P_2 \cdots P_{\hat{z}}$ is the ReLZ parsing, for a given prefix size ℓ. We are to prove that there are at most $1 + 2\sqrt{n}$ indices $j < \hat{z}$ such that $|P_j| < \frac{1}{4}\log_\sigma n$ and $|P_{j+1}| < \frac{1}{4}\log_\sigma n$. This will imply that every phrase of length less than $\frac{1}{4}\log_\sigma n$ is followed by a phrase of length at least $\frac{1}{4}\log_\sigma n$, except for at most $2 + 2\sqrt{n}$ exceptions ($1 + 2\sqrt{n}$ plus the last phrase). Therefore, the total number of phrases is at most $2 + 2\sqrt{n} + \frac{2n}{(1/4)\log_\sigma n} = 2 + 2\sqrt{n} + \frac{8n}{\log_\sigma n}$; the term $2 + 2\sqrt{n}$ is upperbounded by $\frac{n}{\log_\sigma n}$ since $n > 512$, and thus, the total number of phrases is at most $\frac{9n}{\log_\sigma n}$ as required.

It remains to prove that there are at most $1 + 2\sqrt{n}$ pairs of "short" phrases P_j, P_{j+1}. First, observe that any two equal phrases of the LZ parsing of the prefix $T[1,\ell]$ are followed by distinct symbols, except, possibly, for the last phrase. Hence, there are at most $1 + \sum_{k=1}^{\lfloor \frac{1}{4}\log_\sigma n\rfloor} \sigma^{k+1} \le 1 + \sigma^2 \sigma^{\frac{1}{4}\log_\sigma n} \le 1 + n^{2/9} n^{1/4} \le 1 + \sqrt{n}$ phrases of length less than $\frac{1}{4}\log_\sigma n$ in the LZ parsing of $T[1,\ell]$. Further, there cannot be two distinct indices $j < j' < \hat{z}$ such that $P_j = P_{j'}$, $P_{j+1} = P_{j'+1}$, and $|P_1 P_2 \cdots P_{j-1}| \ge \ell$ (i.e., P_j and $P_{j'}$ both are inside the $T[\ell+1,n]$ part of T): indeed, the RLZ step of ReLZ necessarily parses the substrings P_j, P_{j+1} and $P_{j'}$, $P_{j'+1}$ equally, and then, the LZ step of ReLZ should have realized during the parsing of $P_{j'}P_{j'+1}$ that this string occurred previously in $P_j P_{j+1}$ and it should have generated a new phrase comprising $P_{j'}P_{j'+1}$. Therefore, there are at most $\sigma^{\frac{1}{4}\log_\sigma n}\sigma^{\frac{1}{4}\log_\sigma n} = \sqrt{n}$ indices $j < \hat{z}$ such that P_j and P_{j+1} both are "short" and $P_j P_{j+1}$ is inside $T[\ell+1,n]$. In total, we have at most $1 + 2\sqrt{n}$ phrases P_j such that $|P_j| < \frac{1}{4}\log_\sigma n$ and $|P_{j+1}| < \frac{1}{4}\log_\sigma n$. □

Lemmas 3 and 4 immediately imply the following theorem.

Theorem 1 *Given a string $T[1,n]$ over the alphabet $\{1,2,\dots,\sigma\}$ with $\sigma \le O(n)$, the rightmost encoding of any ReLZ parsing of T in which every pair (d_j,ℓ_j) corresponding to a copying phrase takes $\log d_j + O(1+\log\log d_j + \log \ell_j)$ bits occupies $nH_k + o(n\log\sigma)$ bits, for $k = o(\log_\sigma n)$.*

For LZ, it is not necessary to use neither the rightmost encoding nor less than $\log n$ bits for the d_j components of pairs in order to achieve the kth order empirical entropy with $k = o(\log_\sigma n)$. In view of this, the natural question is whether the ReLZ really requires these two assumptions of Theorem 1. The following example shows that indeed the assumptions cannot be simply removed.

Example 1 Fix an integer $b \ge 3$. Our example is a string of length $n = b2^{2b} + 2^b$ over the alphabet $\{0,1,2\}$. Denote by $a_1, a_2, \dots, a_{2^b}$ all possible binary strings of length b. Put $A = a_1 2 a_2 2 \cdots a_{2^b} 2$ ($a_1, a_2, \dots, a_{2^b}$ separated by 2s). The example string is $T = AB_1B_2\cdots B_{2^b-1}$, where each string B_h is the concatenation of $a_1, a_2, \dots, a_{2^b}$ in a certain order such that every pair of distinct strings a_i and a_j can be concatenated in $B_1B_2\cdots B_{2^b-1}$ at most once. More precisely, we have $2^b - 1$ permutations π_h of the set $\{1,2,\dots,2^b\}$, for $1 \le h < 2^b$, such that $B_h = a_{\pi_h(1)}a_{\pi_h(2)}\cdots a_{\pi_h(2^b)}$ and, for every integers i and j with $1 \le i < j \le 2^b$, at most one h satisfies $\pi_h(2^b) = i$ and $\pi_{h+1}(1) = j$, or $\pi_h(k) = i$ and $\pi_h(k+1) = j$, for some $k < 2^b$.

Let us show that the permutations π_h can be constructed from a decomposition of the complete directed graph $K^*_{2^b}$ with 2^b vertices into $2^b - 1$ disjoint Hamiltonian directed cycles; Tillson [43] proved that such decomposition always exists for $2^b \ge 8$. (Note that the number of edges in $K^*_{2^b}$ is $2^{2b} - 2^b$ and every Hamiltonian cycle contains 2^b edges, so $2^b - 1$ is the maximal number of disjoint cycles.) Denote the vertices of $K^*_{2^b}$ by $1,2,\dots,2^b$. Every Hamiltonian cycle naturally induces 2^b permutations π: we arbitrarily choose $\pi(1)$ and then, for $k > 1$, put $\pi(k)$ equal to the vertex number following $\pi(k-1)$ in the cycle. Since the cycles are disjoint, any two distinct numbers i and j cannot occur in this order in two permutations corresponding to different cycles, i.e., $\pi_h(k) = i$ and $\pi_h(k+1) = j$, for some k, can happen at most in one h; further, we put $\pi_1(1) = 1$ and, for $h > 1$, we assign to $\pi_h(1)$ the vertex number following $\pi_{h-1}(2^b)$ in the cycle corresponding to π_{h-1}, so that $\pi_{h-1}(2^b) = i$ and $\pi_h(1) = j$, for fixed i and j, can happen in at most one h.

Put $\ell = |A|$, the parameter of ReLZ. Clearly, the RLZ step of ReLZ parses $B_1B_2\cdots B_{2^b-1}$ into $2^b(2^b-1)$ phrases of length b. By construction, all equal phrases in the parsing are followed by distinct phrases. Therefore, the LZ step of ReLZ does not reduce the number of phrases. Suppose that the source of every copying phrase is in A (so, we assume that the encoding is not rightmost) and we spend at least $\log d_j$ bits to encode each pair (d_j,ℓ_j) corresponding to a copying phrase. Therefore, the encoding overall occupies at least $\sum_{i=1}^{2^b(2^b-1)} \log(ib)$ bits, which can be lowerbounded by $\sum_{i=1}^{2^{2b}-2^b} \log i = \log((2^{2b}-2^b)!) = (2^{2b}-2^b)\log(2^{2b}-2^b) - O(2^{2b})$. Recall that $n = b2^{2b} + 2^b$

and, hence, $b = \Theta(\log n)$, $2^{2b} = o(n)$, and $2^b \log(2^{2b} - 2b) = o(n)$. Thus, $(2^{2b} - 2^b)\log(2^{2b} - 2^b) - O(2^{2b}) \geq 2^{2b}\log(2^{2b} - 2^b) - o(n)$. By the inequality $\log(x - d) \geq \log x - \frac{d \log e}{x-d}$, the latter is lowerbounded by $2^{2b}\log(2^{2b}) - O(2^{2b}2^b/(2^{2b} - 2^b)) - o(n) = 2b2^{2b} - o(n) = 2n - o(n)$. On the other hand, we obviously have $H_0(T) \approx 1$ and, thus, $nH_0(T) = n - o(n)$. Therefore, the non-rightmost encoding, which forced us to use at least $\sim\log n$ bits for many pairs (d_j, ℓ_j), does not achieve the zeroth empirical entropy of T.

5 Lower Bound

We have not been able to upper bound the number of phrases $\hat{z}$ resulting from ReLZ in terms of the optimal number z of phrases produced by the LZ parsing of T. Note that, in the extreme cases $\ell = n$ and $\ell = 0$, we have $\hat{z} = z$, but these are not useful choices: in the former case we apply $LZ(T)$ in the first phase and in the latter case we apply $LZ(T')$, with $T' \approx T$, in the second phase. In this section, we obtain the following lower bound.

Theorem 2 *There is an infinite family of strings over the alphabet* $\{0, 1, 2\}$ *such that, for each family string* $T[1, n]$*, the number of phrases in its ReLZ parse (with an appropriate parameter* $\ell = o(n)$*) and its LZ parse—respectively,* $\hat{z}$ *and* z*—are related as* $\hat{z} = \Omega(z \log n)$*.*

Proof The family contains, for each even positive integer b, a string T of length $\Theta(b^2 2^b)$ built as follows. Let A be the concatenation of all length-b binary strings in the lexicographic order, separated by the special symbol 2 and with 2 in the end. Let S be the concatenation of all length-b binary strings in the lexicographic order. (E.g., $A = 002012102112$ and $S = 00011011$ for $b = 2$.) Finally, let S_i be S cyclically shifted to the left i times, i.e., $S_i = S[i + 1, |S|] \cdot S[1, i]$. Then, put $T = AS_1S_2 \cdots S_{\frac{b}{2}}$ and we use $\ell = |A|$ as a parameter for ReLZ. So $n = |T| = \Theta(b^2 2^b)$ and $\log n = \Theta(b)$. We are to prove that $z = |LZ(T)| = O(2^b)$ and $\hat{z} = |\mathrm{ReLZ}(T, \ell)| = \Omega(b2^b)$, which will imply $\hat{z} = \Omega(z \log n)$, thus concluding the proof.

By [49, Th. 1], the LZ parse has the smallest number of phrases among all LZ-like parses of T. Therefore, to show that $z = O(2^b)$, it suffices to describe an LZ-like parse of T with $O(2^b)$ phrases. Indeed, the prefix A can be parsed into $O(2^b)$ phrases as follows: all symbols 2 form phrases of length one; the first length-b substring $00\cdots0$ can be parsed into b literal phrases 0; every subsequent binary length-b substring $a_1a_2\cdots a_b$ with $a_k = 1$ and $a_{k+1} = a_{k+2} = \cdots = a_b = 0$, for some $k \in \{1, 2, \ldots, b\}$, can be parsed into the copying phrase $a_1a_2\cdots a_{k-1}$ (which must be a prefix of the previous length-b binary substring $a_1a_2\cdots a_{k-1}011\cdots1$, due to the lexicographic order in A), the literal phrase 1, and the copying phrase $a_{k+1}a_{k+2}\cdots a_b = 00\cdots0$. The string S_1 can be analogously parsed into $O(2^b)$ phrases. Each S_i, $i > 2$, can be expressed as two phrases that point to S_1. Thus, we obtain $z \leq |LZ(A)| + |LZ(S_1)| + 2(b/2 - 1) = O(2^b)$.

Now consider $\hat{z}$. The first phase of ReLZ(T, ℓ) parses T into phrases whose sources are restricted to be within $T[1, \ell] = A$. Therefore, it is clear that, for any $i \in \{1, 2, \ldots, \frac{b}{2}\}$, S_i will be parsed into 2^b strings of length b, because every length-b string is in A separated by 2s. In what follows we show that the second phase of ReLZ cannot further reduce the number of phrases and, hence, $\hat{z} \geq \frac{b}{2} 2^b = \Omega(b2^b)$ as required.

Let us consider S_i and S_j, for some $i < j$, and let us denote their parsings by $R_1, R_2, \ldots, R_{2^b}$ and $R'_1, R'_2, \ldots, R'_{2^b}$, respectively. Suppose that there are indices k and h such that $R_k = R'_h$. We are to prove that $R_{k+1} \neq R'_{h+1}$ (assuming R_{k+1} is the length-b prefix of S_{i+1} if $k = 2^b$, and analogously for $h = 2^b$). This will imply that all phrases produced by the second phase of ReLZ on the string of metasymbols are of length one.

Consider the case $k < 2^b$ and $h < 2^b$. Let us interpret the bitstrings of length b as numbers so that the most and the least significant bits are indexed by 1 and b, respectively;[2] e.g., in the string 01, for $b = 2$, the least significant bit is the second symbol and equals 1. In this way we can see $S = Q_1 Q_2 \cdots Q_{2^b}$, where $|Q_1| = \cdots = |Q_{2^b}| = b$, as generated by adding 1 to the previous bitstring, starting from $Q_1 = 00 \cdots 0$. Now, the $(b - i)$th symbols of R_k and R_{k+1} are different since they correspond to the lowest bit in $Q_1, Q_2, \ldots, Q_{2^b}$ (thus, the $(b-i)$th symbol alternates in $R_1, \ldots, R_{2^b}$, starting from 0). Suppose that the $(b - i)$th symbols of R'_h and R'_{h+1} also differ (otherwise our claim is trivially true). Since $0 < i < j$, this implies that the symbols $b, b-1, \ldots, b-i+1$ in R'_h and $1, 2, \ldots, b-j$ in R'_{h+1} all are equal to 1 (this cascade of ones triggers the change in the $(b - i)$th symbol of R'_{h+1}), the symbols $b, b - 1, \ldots, b - i + 1$ in R'_{h+1} equal 0 (as a result of the "collapse" of the cascade), and the $(b - j)$th symbol in R'_h equals 0 (since $(b - j)$th symbols alternate in $R'_1, \ldots, R'_{2^b}$ and the $(b - j)$th symbol in R'_{h+1} equals 1 as a part of the cascade).

In the following example $b = 12$, $i = 4$, $j = 8$, $\diamond$ denotes irrelevant symbols (not necessarily equal), x and $\overline{x}$ denote the flipped $(b-i)$th symbol, the $(b-j)$th symbol is underlined:

$$\begin{aligned} R'_h &= \diamond\diamond\diamond\underline{0}\diamond\diamond\diamond x1111, \\ R'_{h+1} &= 111\underline{1}\diamond\diamond\diamond\overline{x}0000. \end{aligned}$$

When we transform $R_k = R'_h$ to R_{k+1}, we "add" 1 to the bit corresponding to the $(b-i)$th symbol of R_k and the zero at position $b-j$ will stop carrying the 1, so that we necessarily have zero among the symbols $b - i, b - i - 1, \ldots, b - j$ of R_{k+1} (in fact, one can show that they all are zeros except for $b - j$). Thus, the next "addition" of 1 to the $(b - i)$th symbol of R_{k+1} cannot carry farther than the $(b - j)$th symbol and so the symbols $b, b - 1, \ldots, b - i + 1$ will remain equal to 1 in R_{k+1} whilst in R'_{h+1} they are all zeros. Therefore, $R'_{h+1} \neq R_{k+1}$.

In the case $k = 2^b, R_k = 11 \cdots 100 \cdots 0$, with $b - i$ ones, is followed by $R_{k+1} = 00 \cdots 0$, with b zeros. But, since $R'_h = R_k$ and $i < j$, we have $R'_{h+1} = 00 \cdots 011 \cdots 100 \cdots 0$, with $j - i$ ones, after "adding" 1 to the $(b - j)$th symbol of R'_h. The case $h = 2^b$ is analogous. □

[2] To conform with the indexation scheme used throughout the paper, we do not follow the standard practice to index the least significant bit as zeroth.

6 Implementation

To build RLZ$_{pref}$, we first compute $LZ(T[1,\ell])$ and then $RLZ(T[\ell{+}1,n],T[1,\ell])$. For both of them, we utilize the suffix array of $T[1,\ell]$, which is constructed using the algorithm `libdivsufsort` [48,18]. To compute $LZ(T[1,\ell])$, we use the KKP3 algorithm [20]. To compute $RLZ(T[\ell+1,n],T[1,\ell])$, we scan $T[\ell+1,n]$ looking for the longest match in $T[1,\ell]$ by the standard suffix array based pattern matching.

The output phrases are encoded as pairs of integers: each pair (p_j,ℓ_j) represents the position, p_j, of the source for the phrase and the length, ℓ_j, of the phrase (note that this is in contrast to the "distance-length" pairs (d_j,ℓ_j) that we had for encodings). We then map the output into a sequence of numbers using $2\lceil\log\ell\rceil$-bit integers with $\lceil\log\ell\rceil$ bits for each pair component. This is possible because we enforce that our reference size is $\ell \geq \sigma$.

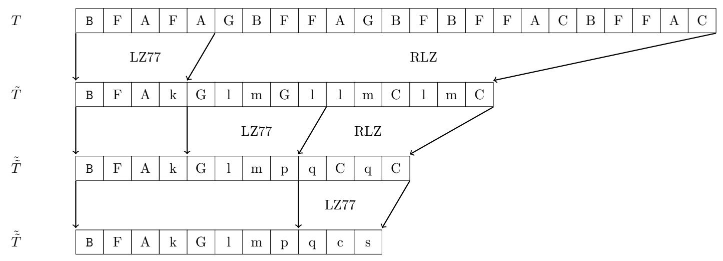
Finally, we compute the LZ parse using a version of KKP3 for large alphabets, relying on a suffix array construction algorithm for large alphabets [30, 18]. We then remap the output of LZ to point to positions in T as described.

6.1 A recursive variant

When the input is too big compared to the available RAM, it is possible that after the first compression step, RLZ$_{pref}$, the resulting parse is still too big to fit in memory, and therefore it is still not possible to compute its LZ parse efficiently. To overcome this issue in practice, we propose a recursive variant, which takes as input the amount of available memory. The first step remains the same, but in the second step we make a recursive call to ReLZ, ignoring the phrases that were already parsed with LZ, and using the longest possible ℓ value for the given amount of RAM. This recursive process continues until the LZ parse can be computed in memory. It is also possible to give an additional parameter that limits the number of recursive calls. We use the recursive version only in the last set of experiments when comparing with other LZ parsers in Subsection 7.4

6.2 A better mapping

When the recursive approach is used we need a better mapping from pairs of integers into integers: the simple approach described above requires $2\log\ell$ bits for the alphabet after the first iteration, but in the following iterations the assumption $\sigma \leq \ell$ may not hold anymore and the amount of bits required to store the first values may increase at each iteration. We propose a simple mapping that overcomes this problem. Let σ_i be the size of the alphabet used by the metasymbols after the ith iteration. To encode the metasymbols of the $(i+1)$-iteration we use first a flag bit to indicate whether the phrase is literal or copying. If the flag is 0, then it is a literal phrase $(c,0)$ and $\log\sigma_i$ bits are



T
B F A F A G B F F A G B F B F F A C B F F A C
LZ77
RLZ
$\tilde{T}$
B F A k G l m G l l m C l m C
LZ77
RLZ
$\tilde{\tilde{T}}$
B F A k G l m p q C q C
LZ77
$\tilde{\tilde{\tilde{T}}}$
B F A k G l m p q c s

Fig. 2 Example of the recursive ReLZ approach, assuming that the available memory limits the computation of LZ to sequences of length 5. The figure only shows the recursive parsing. The rewriting of the phrases proceeds later, bottom up, in a similar fashion as depicted in Figure 1.

used to store the c value. If the flag is 1, then it is a copying phrase (p_i, ℓ_i) and then $2 \log \ell$ bits are used to store the numbers. In this way, after each iteration the number of bits required to store the metasymbols increases only by 1.

Table 1 Collections used for the experiments, some basics statistics, and a brief description of their source. The first group includes medium-sized collections, from 45 to 202 MiB, while the second group consist of large collections, from 22 to 64GiB. Each group has both regular collections and highly repetitive collections, attested by the average phrase length n/z.

Name	σ	n	n/z	Type	Source
English	225	200 MiB	15	English text	Pizzachili
Sources	230	202 MiB	18	Source code	Pizzachili
Influenza	15	148 MiB	201	Genomes	Pizzachili
Leaders	89	45 MiB	267	English text	Pizzachili
Wiki	215	24 GiB	90	Web pages	Wikipedia dumps
Kernel	229	64 GiB	2439	Source code	Linux Kernel
CereHR	5	22 GiB	3746	Genomes	Pizzachili

We implemented ReLZ in C++ and the source code is available under GPLv3 license in `https://gitlab.com/dvalenzu/ReLZ`. The implementation allows the user to set the value of ℓ or, alternatively, to provide the maximum amount of RAM to be used. Additionally, scripts to reproduce our experiments are available at `https://gitlab.com/dvalenzu/ReLZ_experiments`. For the experimental evaluation, we used collections of different sizes and kinds. They are listed in Table 1 with their main properties. The experiments were run on a desktop computer equipped with a Intel(R) Core(TM) i5-7500 CPU, with 4 cores, 3.60GHz and 16GB of RAM.

7 Experimental evaluation

7.1 Entropy coding

First we compare the encoded size of ReLZ with the k-order empirical entropy, and also with the encoded size of LZ77. For both ReLZ and LZ77 we used Elias-gamma codes. The results are presented in table 2.

We observe that in the small and low-repetition collections (English and source) ReLZ requires some extra space than H_k for higher values of k. This can be attributed to the $o()$ term in our analysis. Also we observe the same behavior for LZ77. As expected, for the highly repetitive collections, both ReLZ and LZ77 use less space than the entropy. This is due to the known fact that for highly repetitive collections, z is a better measurement of the compressibility than the empirical entropy. Therefore, in the following sections, we proceed to study empirically how does ReLZ compare to LZ77 in terms of number of phrases produced by the parsers.

Table 2 Empirical entropy of order k of our collections for $k = 1, 2, \ldots 6$; and encoded size of ReLZ and LZ77. All values are expressed as bits per character (bpc).

	Entropy H_k							ReLZ(bpc); $\ell =$		LZ(bpc)
Name	H_0	H_1	H_2	H_3	H_4	H_5	H_6	10MB	50MB	
English	4.52	3.62	2.94	2.42	2.06	1.83	1.67	3.53	3.02	2.65
Sources	5.46	4.07	3.10	2.33	1.85	1.51	1.24	2.97	2.64	1.94
Influenza	1.97	1.93	1.92	1.92	1.91	1.87	1.76	0.29	0.24	0.20
Leaders	3.47	1.95	1.38	0.93	0.60	0.40	0.32	0.15	0.13	0.13
Wiki	5.27	3.86	2.35	1.49	1.08	0.86	0.71	0.79	0.80	0.56
Kernel	5.58	4.14	3.16	2.39	1.92	1.58	1.32	0.02	0.02	0.018
CereHR	2.19	1.81	1.81	1.80	1.80	1.80	1.80	0.02	0.02	0.013

7.2 Effect of Reference Sizes

We first study how the size of the prefix used as a reference influences the number of phrases produced by RLZ$_{pref}$ and ReLZ. These experiments are carried out only using the medium-sized collections, so that we can run ReLZ using arbitrarily large prefixes as references and without recursions. We ran both algorithms using different values of $\ell = n/10, 2n/10, \ldots, n$.

The results are presented in Figure 3. By design, both algorithms behave as LZ when $\ell = n$. RLZ$_{pref}$ starts far off from LZ and its convergence is not smooth but "stepped". The reason is that at some point, by increasing ℓ, the reference captures a new sequence that has many repetitions that were not well compressed for smaller values of ℓ. Thus RLZ$_{pref}$ is very dependent on the choice of the reference. ReLZ, in contrast, is more robust since the second pass of LZ does capture much of those global repetitions. This results in ReLZ

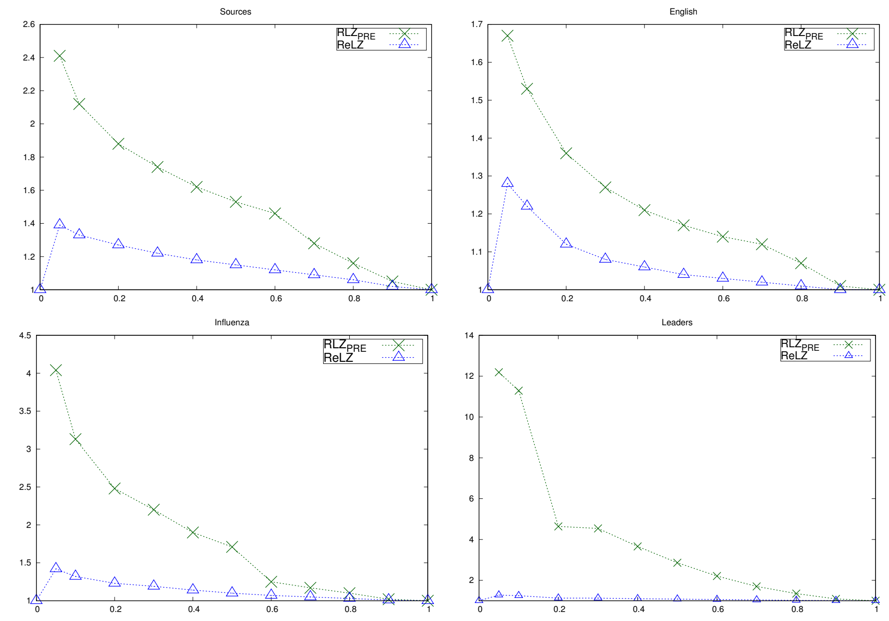


Fig. 3 Performance of RLZ$_{pref}$ (green) and ReLZ (blue) for different prefix-reference sizes on medium-sized inputs. The y-axis shows the approximation ratio $\hat{z}/z$. The x-axis shows ℓ/n, the size of the prefix-reference expressed as a fraction of the input size.

being very close to LZ even for $\ell = n/10$, particularly in the highly repetitive collections.

7.3 Reference Construction

As discussed in Section 3, the idea of a second compression stage applied to the phrases can be applied not only when the reference is a prefix, but also when an external reference is used. This allows us to study variants of ReLZ combined with different strategies to build the reference that aim for a better compression in the first stage.

In this section we experimentally compare the following approaches:

`PREFIX`: Original version using a prefix as a reference.
`RANDOM`: An external reference is built as a concatenation of random samples of the collection [19,16].
`PRUNE`: A recent method [33] that takes random samples of the collections and performs some pruning of redundant parts to construct a better reference.

An important caveat is that methods using an external reference also need to account for the reference size in the compressed representation because the reference is needed to recover the output. For each construction method, we measure the number of phrases produced for the string “reference + text” (only

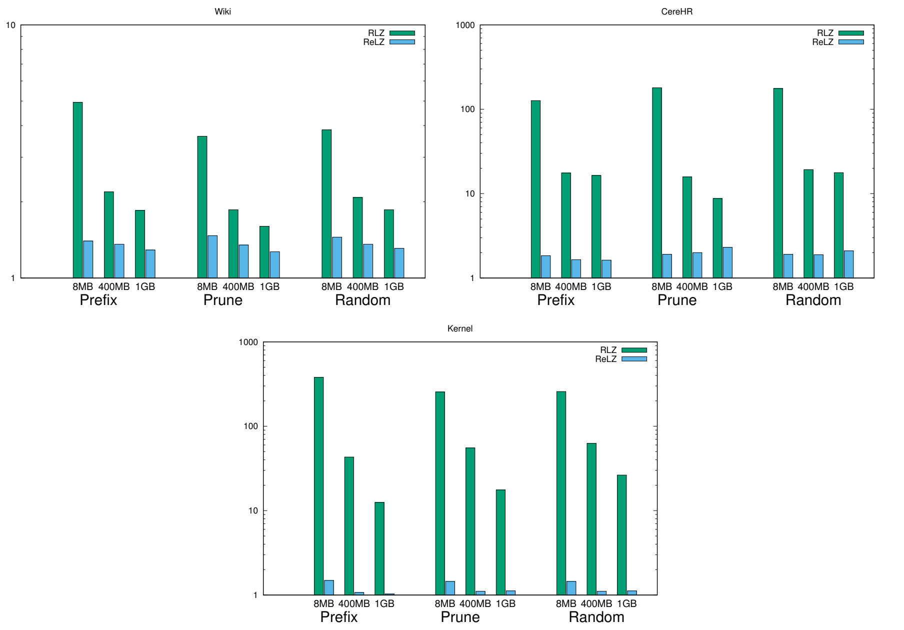


Fig. 4 Approximation ratio $\hat{z}/z$ for different methods to construct the reference and different reference lengths: in green the results after RLZ$_{pref}$, and in blue after ReLZ. Note that the highly repetitive collections (CereHR and Kernel) use logarithmic scale.

"text" for the method `PREFIX`) by the first stage (RLZ$_{pref}$ with prefix equal to the reference) and by the second stage (LZ on metasymbols corresponding to the phrases), using three reference sizes: 8MB, 400MB, and 1GB. We compare the numbers to z, the number of phrases in the LZ parsing of the plain text. This experiment was performed on the large collections and the results are presented in Figure 4.

We observe that the second stage of ReLZ reduces the number of phrases dramatically, regardless of the reference construction method. ReLZ with the original method `PREFIX` achieves the best ratios as it does not need to account for the external reference. Depending on the reference size, the approximation ratio in Wiki ranges between 1.4 and 1.29, in CereHR between 1.84 and 1.63, and in Kernel between 1.49 and 1.03.

Additionally, we observe that although `PRUNE` can improve the results of the RLZ$_{pref}$ stage, after the second stage the improvements do not compensate for the need to keep an external reference. This is particularly clear for the largest reference in our experiments.

7.4 Lempel–Ziv Parsers

In this section we compare the performance and scalability of ReLZ against other Lempel–Ziv parsers that can also run in small memory (this time, using the recursive version of ReLZ).

`EMLZ` [21]: External-memory version of the exact LZ algorithm, with memory usage limit set to 4GB.
`LZ-End` [22]: An LZ-like parsing that gets close to LZ in practice.
`ORLBWT` [3]: Computes the exact LZ parsing via online computation of the RLBWT using small memory.
`RLZ`$_{PRE}$: Our RLZ$_{pref}$ algorithm (Section 3), with memory usage limit set to 4GB.
`ReLZ`: Our ReLZ algorithm (Section 3), with memory usage limit set to 4GB.

To see how well the algorithms scale with larger inputs, we took prefixes of different sizes of all the large collections and ran all the parsers on them. We measured the running time of all of the algorithms and, for the algorithms that do not compute the exact LZ parsing, we also measured the approximation ratio $\hat{z}/z$. The results are presented in Figure 5.

Figure 5 (left) shows that ReLZ is much faster than all the previous methods and also that the speed is almost unaffected when processing larger inputs. Figure 5 (right) shows that the approximation ratio of ReLZ is affected very mildly as the input size grows, especially in the highly repetitive collections. For the normal collections, the approximation factor is more affected but it still remains below 2.

7.5 Compression ratio

In this section we study the compression ratio of ReLZ. We store the *pos* and *len* values in separate files, encoding them using a modern implementation of PFOR codes [31] in combination with a fast entropy coder [8]. We compare against state of the art compressors (LZMA, Brotli) and also agains a very recent RLZ compressor (RLZ-store). We measure compression ratios, compression times and decompression times of these tools in the large collections, whose size exceeds the available RAM of the system.

The results are shown in Figure 6. In the normal collection (Wiki) the performance of ReLZ is competitive with the state of the art compressors. In the highly repetitive collections (Cere, Kernel) ReLZ gives the best compression ratios, with very similar compression times and competitive decompression times.

Additionally, we run a comparisson again GDC2 and FRESCO. Both tools are designed to compress a collection of files, using one (or more) as a reference, and perform referential compression plus second order compression. GDC2 is specifically designed to compress collections of genomes in FASTA format, and it exploits known facts about genomes collections (e.g. an important amount

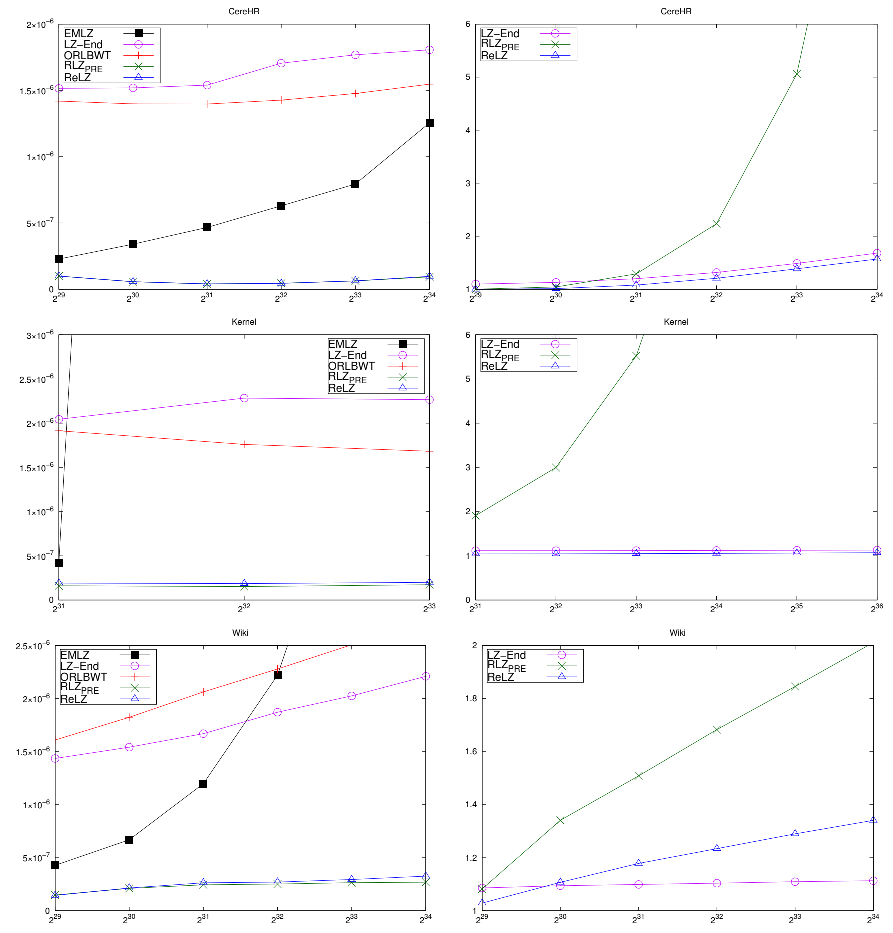


Fig. 5 Performance of different LZ parsers in the large collections. The x axis is the size of the input: increasingly larger prefixes of a given collection. Plots on the left show the running time in seconds per MiB. Plots on the right show the approximation ratio $\hat{z}/z$.

of the variations are changes in a single character). For this, we use a 90GB collection comprising 2001 different versions of chromosome 21. As expected, GDC2 was the dominant tool, with a compression ratio of 0.00020, compression time of 15 minutes and decompression time of 15 minutes. ReLZ compression ratio was 0.00047, compression time was 49 minutes and decompression time was 50 minutes. We stopped FRESCO execution after 8 hours, when it had processed slightly more than half of the collection.

Acknowledgements This work started during Shonan Meeting 126 “Computation over Compressed Structured Data”. Funded in part by EU’s Horizon 2020 research and innovation programme under Marie Skłodowska-Curie grant agreement No 690941 (project BIRDS).

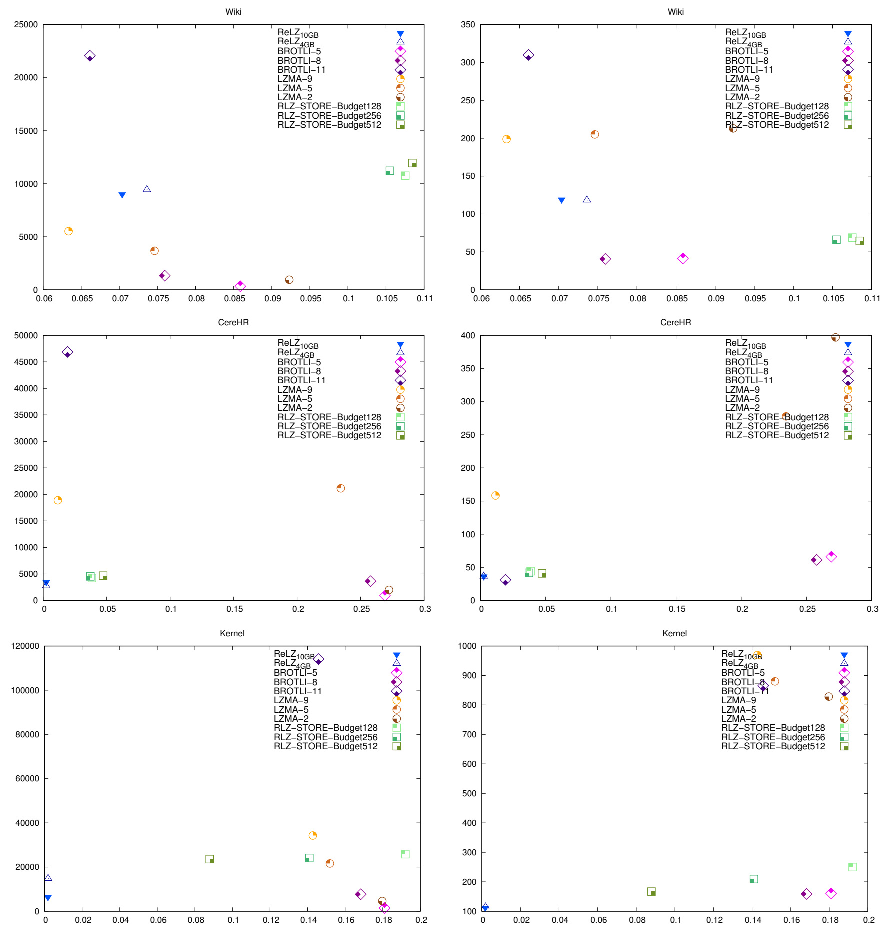


Fig. 6 Compression results for large collections. The x-axis is the ratio between output and input, and the y-axis is the total compression time in seconds.

References

1. Alakuijala, J., Farruggia, A., Ferragina, P., Kliuchnikov, E., Obryk, R., Szabadka, Z., Vandevenne, L.: Brotli: A general-purpose data compressor. ACM Transactions on Information Systems **37**(1), 4 (2018). DOI 10.1145/3231935
2. Amir, A., Landau, G.M., Ukkonen, E.: Online timestamped text indexing. Information Processing Letters **82**(5), 253–259 (2002). DOI 10.1016/S0020-0190(01)00275-7
3. Bannai, H., Gagie, T., I, T.: Online LZ77 parsing and matching statistics with RLBWTs. In: Proc. CPM 2018, *LIPIcs*, vol. 105, pp. 7:1–7:12. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik (2018). DOI 10.4230/LIPIcs.CPM.2018.7
4. Belazzougui, D., Puglisi, S.J.: Range predecessor and Lempel–Ziv parsing. In: Proc. SODA 2016, pp. 2053–2071. SIAM (2016). DOI 10.1137/1.9781611974331.ch143
5. Bille, P., Cording, P.H., Fischer, J., Gørtz, I.L.: Lempel–Ziv compression in a sliding window. In: Proc. CPM 2017, *LIPIcs*, vol. 78. Schloss Dagstuhl-Leibniz-Zentrum für Informatik (2017). DOI 10.4230/LIPIcs.CPM.2017.15

6. Deorowicz, S., Danek, A., Niemiec, M.: GDC 2: Compression of large collections of genomes. Scientific reports **5**, 11565 (2015). DOI doi.org/10.1038/srep11565
7. Deorowicz, S., Grabowski, S.: Robust relative compression of genomes with random access. Bioinformatics **27**(21), 2979–2986 (2011). DOI 10.1093/bioinformatics/btr505
8. Duda, J.: Asymmetric numeral systems as close to capacity low state entropy coders. CoRR **abs/1311.2540** (2013). URL `http://arxiv.org/abs/1311.2540`
9. Elias, P.: Universal codeword sets and representations of the integers. IEEE Transactions on Information Theory **21**(2), 194–203 (1975). DOI 10.1109/TIT.1975.1055349
10. Ferragina, P., Nitto, I., Venturini, R.: On the bit-complexity of Lempel–Ziv compression. SIAM Journal on Computing **42**(4), 1521–1541 (2013). DOI 10.1137/120869511
11. Fischer, J., Gagie, T., Gawrychowski, P., Kociumaka, T.: Approximating LZ77 via small-space multiple-pattern matching. In: Proc. ESA 2015, *LNCS*, vol. 9294, pp. 533–544. Springer (2015). DOI 10.1007/978-3-662-48350-3_45
12. Gagie, T.: Large alphabets and incompressibility. Information Processing Letters **99**(6), 246–251 (2006). DOI 10.1016/j.ipl.2006.04.008
13. Gagie, T., Manzini, G.: Space-conscious compression. In: Proc. MFCS 2007, *LNCS*, vol. 4708, pp. 206–217. Springer (2007). DOI 10.1007/978-3-540-74456-6_20
14. Gagie, T., Navarro, G., Prezza, N.: On the approximation ratio of Lempel–Ziv parsing. In: Proc. LATIN 2018, *LNCS*, vol. 10807, pp. 490–503. Springer (2018). DOI 10.1007/978-3-319-77404-6_36
15. Gagie, T., Navarro, G., Prezza, N.: Optimal-time text indexing in BWT-runs bounded space. In: Proc. SODA 2018, pp. 1459–1477. SIAM (2018). DOI 10.1137/1.9781611975031.96
16. Gagie, T., Puglisi, S.J., Valenzuela, D.: Analyzing relative Lempel–Ziv reference construction. In: Proc. SPIRE 2016, *LNCS*, vol. 9954, pp. 160–165. Springer (2016). DOI 10.1007/978-3-319-46049-9_16
17. Gańczorz, M.: Entropy bounds for grammar compression. CoRR **abs/1804.08547** (2018). URL `http://arxiv.org/abs/1804.08547`
18. Gog, S., Beller, T., Moffat, A., Petri, M.: From theory to practice: Plug and play with succinct data structures. In: Proc. SEA 2014, *LNCS*, vol. 8504, pp. 326–337. Springer (2014). DOI 10.1007/978-3-319-07959-2_28
19. Hoobin, C., Puglisi, S.J., Zobel, J.: Relative Lempel–Ziv factorization for efficient storage and retrieval of web collections. Proc. the VLDB Endowment **5**(3), 265–273 (2011). DOI 10.14778/2078331.2078341
20. Kärkkäinen, J., Kempa, D., Puglisi, S.J.: Linear time Lempel–Ziv factorization: Simple, fast, small. In: Proc. CPM 2013, *LNCS*, vol. 7922, pp. 189–200. Springer (2013). DOI 10.1007/978-3-642-38905-4_19
21. Karkkainen, J., Kempa, D., Puglisi, S.J.: Lempel–Ziv parsing in external memory. In: Proc. DCC 2014, pp. 153–162. IEEE (2014). DOI 10.1109/DCC.2014.78
22. Kempa, D., Kosolobov, D.: LZ-End parsing in compressed space. In: Proc. DCC 2017, pp. 350–359. IEEE (2017). DOI 10.1109/DCC.2017.73
23. Kempa, D., Prezza, N.: At the roots of dictionary compression: string attractors. In: Proc. STOC 2018, pp. 827–840. ACM (2018). DOI 10.1145/3188745.3188814
24. Kosaraju, S.R., Manzini, G.: Compression of low entropy strings with Lempel–Ziv algorithms. SIAM Journal on Computing **29**(3), 893–911 (1999). DOI 10.1137/S0097539797331105
25. Kosolobov, D.: Relations between greedy and bit-optimal LZ77 encodings. In: Proc. STACS 2018, *LIPIcs*, vol. 96, pp. 46:1–46:14. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik (2018). DOI 10.4230/LIPIcs.STACS.2018.46
26. Kreft, S., Navarro, G.: LZ77-like compression with fast random access. In: Proc. DCC 2010, pp. 239–248. IEEE (2010). DOI 10.1109/DCC.2010.29
27. Kuruppu, S., Puglisi, S.J., Zobel, J.: Relative Lempel–Ziv compression of genomes for large-scale storage and retrieval. In: Proc. SPIRE 2010, *LNCS*, vol. 6393, pp. 201–206. Springer (2010). DOI 10.1007/978-3-642-16321-0_20
28. Kuruppu, S., Puglisi, S.J., Zobel, J.: Optimized relative Lempel–Ziv compression of genomes. In: Australasian Computer Science Conference, pp. 91–98. Australian Computer Society, Inc. (2011)
29. Larsson, N.J.: Most recent match queries in on-line suffix trees. In: Proc. CPM 2014, *LNCS*, vol. 8486, pp. 252–261 (2014). DOI 10.1007/978-3-319-07566-2_26

30. Larsson, N.J., Sadakane, K.: Faster suffix sorting. Theoretical Computer Science **387**(3), 258–272 (2007). DOI 10.1016/j.tcs.2007.07.017
31. Lemire, D., Boytsov, L.: Decoding billions of integers per second through vectorization. Software: Practice and Experience **45**(1), 1–29 (2015)
32. Levenshtein, V.I.: On the redundancy and delay of decodable coding of natural numbers. Systems Theory Research **20**, 149–155 (1968)
33. Liao, K., Petri, M., Moffat, A., Wirth, A.: Effective construction of relative Lempel–Ziv dictionaries. In: Proc. WWW 2016, pp. 807–816. International World Wide Web Conferences Steering Committee (2016). DOI 10.1145/2872427.2883042
34. Mäkinen, V., Navarro, G.: Compressed full-text indexes. ACM Computing Surveys **39**(1), 2 (2007). DOI 10.1145/1216370.1216372
35. Manzini, G.: An analysis of the Burrows–Wheeler transform. Journal of the ACM **48**(3), 407–430 (2001). DOI 10.1145/382780.382782
36. Navarro, G.: Indexing highly repetitive collections. In: Proc. IWOCA 2012, *LNCS*, vol. 7643, pp. 274–279 (2012). DOI 10.1007/978-3-642-35926-2_29
37. Ochoa, C., Navarro, G.: RePair and all irreducible grammars are upper bounded by high-order empirical entropy. IEEE Transactions on Information Theory (2018). DOI 10.1109/TIT.2018.2871452
38. Policriti, A., Prezza, N.: Fast online Lempel–Ziv factorization in compressed space. In: Proc. SPIRE 2015, *LNCS*, vol. 9309, pp. 13–20. Springer (2015). DOI 10.1007/978-3-319-23826-5_2
39. Policriti, A., Prezza, N.: LZ77 computation based on the run-length encoded BWT. Algorithmica **80**(7), 1986–2011 (2018). DOI 10.1007/s00453-017-0327-z
40. Puglisi, S.J.: Lempel–Ziv compression. In: Encyclopedia of Algorithms, pp. 1095–1100. Springer (2016)
41. Shields, P.C.: Performance of LZ algorithms on individual sequences. IEEE Transactions on Information Theory **45**(4), 1283–1288 (1999). DOI 10.1109/18.761286
42. Storer, J.A., Szymanski, T.G.: Data compression via textual substitution. Journal of the ACM **29**(4), 928–951 (1982). DOI 10.1145/322344.322346
43. Tillson, T.W.: A hamiltonian decomposition of K^*_{2m}, $2m \geq 8$. Journal of Combinatorial Theory, Series B **29**(1), 68–74 (1980). DOI 10.1016/0095-8956(80)90044-1
44. Valenzuela, D.: CHICO: A compressed hybrid index for repetitive collections. In: Proc. SEA 2016, *LNCS*, vol. 9685, pp. 326–338. Springer (2016). DOI 10.1007/978-3-319-38851-9_22
45. Wandelt, S., Leser, U.: FRESCO: Referential compression of highly similar sequences. IEEE/ACM Transactions on Computational Biology and Bioinformatics **10**(5), 1275–1288 (2013). DOI 10.1109/TCBB.2013.122
46. Wyner, A.J.: The redundancy and distribution of the phrase lengths of the fixed-database Lempel–Ziv algorithm. IEEE Transactions on Information Theory **43**(5), 1452–1464 (1997). DOI 10.1109/18.623144
47. Yann Collet, 2016: Zstandard. Retrieved from: https://facebook.github.io/zstd/. Accessed: 2018-09-17
48. Yuta Mori: libdivsufsort. https://github.com/y-256/libdivsufsort/
49. Ziv, J., Lempel, A.: On the complexity of finite sequences. IEEE Transactions on Information Theory **22**(1), 75–81 (1976). DOI 10.1109/TIT.1976.1055501
50. Ziv, J., Lempel, A.: A universal algorithm for sequential data compression. IEEE Transactions on Information Theory **23**(3), 337–343 (1977). DOI 10.1109/TIT.1977.1055714