SIAM J. Comput.
Vol. 29, No. 3, pp. 893–911

COMPRESSION OF LOW ENTROPY STRINGS WITH LEMPEL–ZIV ALGORITHMS[*]

S. RAO KOSARAJU[†] AND GIOVANNI MANZINI[‡]

Abstract. We compare the compression ratio of the Lempel–Ziv algorithms with the empirical entropy of the input string. This approach makes it possible to analyze the performance of these algorithms without any assumption on the input and to obtain worst case results. We show that in this setting the standard definition of optimal compression algorithm is not satisfactory. In fact, although Lempel–Ziv algorithms are optimal according to the standard definition, there exist families of *low entropy strings* which are not compressed optimally. More precisely, the compression ratio achieved by LZ78 (resp., LZ77) can be much higher than the zeroth order entropy H_0 (resp., the first order entropy H_1).

For this reason we introduce the concept of λ-optimal algorithm. An algorithm is λ-optimal with respect to H_k if, loosely speaking, its compression ratio is asymptotically bounded by λ times the kth order empirical entropy H_k. We prove that LZ78 cannot be λ-optimal with respect to any H_k with $k \geq 0$. Then, we describe a new algorithm which combines LZ78 with run length encoding (RLE) and is 3-optimal with respect to H_0. Finally, we prove that LZ77 is 8-optimal with respect to H_0, and that it cannot be λ-optimal with respect to H_k for any $k \geq 1$.

Key words. data compression, Lempel–Ziv parsing, empirical entropy

AMS subject classifications. 68Q25, 68P20

PII. S0097539797331105

1. Introduction.

The most widely used data compression algorithms are based on the well known procedures LZ77 and LZ78 [23, 24]. These procedures compress the input string by replacing phrases with a pointer to a previous occurrence of the same phrase. This simple scheme achieves a good compression ratio and both coding and decoding can be done on-line very efficiently.

Given the practical relevance of Lempel–Ziv algorithms, many efforts have been done to analyze their performance. In [23] it is shown that LZ77 is optimal for a certain family of sources, and in [24] it is shown that LZ78 achieves asymptotically the best compression ratio attainable by a finite-state compressor. Other results have been obtained assuming that the input string is a random sequence $\{X_i\}_{-\infty}^{+\infty}$ which is stationary, ergodic, and takes values from a finite alphabet. Under these hypotheses, it has been shown that LZ77 and LZ78 are optimal, that is, their compression rate approaches the entropy of the random sequence [16, 20, 24]. Similar results have been obtained also for other variants of the Lempel–Ziv algorithms [2, 10, 18, 19, 22] and considering different probability distributions for the input string [6, 8].

[*] Received by the editors December 8, 1997; accepted for publication (in revised form) December 16, 1998; published electronically December 15, 1999. A preliminary version of this paper appeared in the Proceedings of the Symposium on Compression and Complexity of Sequences (SEQUENCES), Positano, Italy, IEEE Computer Society, Los Alamitos, CA, 1997.

http://www.siam.org/journals/sicomp/29-3/33110.html

[†] Department of Computer Science, Johns Hopkins University, Baltimore, MD (kosaraju@cs.jhu.edu). The work of this author was supported by NSF under grant CCR-9508545 and ARO under grant DAAH04-96-1-0013.

[‡] Dip. Scienze e Tecnologie Avanzate, Università del Piemonte Orientale, I-15100 Alessandria, Italy, and Istituto di Matematica Computazionale, CNR, Pisa, Italy (manzini@mfn.al.unipmn.it). The work of this author was supported by a CNR short term fellowship and by MURST 40% and 60% funds. Part of this work was done when the author was visiting the Department of Computer Science of the Johns Hopkins University, Baltimore, MD.

More recently, a careful analysis of the Lempel–Ziv parsing rules (see, for example, [7, 17, 21]) and the use of renewal theory have made it possible to estimate the *redundancy* of Lempel–Ziv algorithms, that is, the amount by which the expected compression ratio exceeds the entropy of the source (the redundancy is therefore a measure of the speed at which the compression ratio approaches the entropy). In [9, 14] it is shown that for LZ78 and many of its variants the redundancy is $O\big((\log n)^{-1}\big)$, where n is the length of the input string. In [15] it is shown that for two variants of LZ77 the redundancy is $O(\log\log n/\log n)$. Note that for both families of algorithms we also have bounds on the size of the constants hidden in the big-o notation. Finally, in [21] it is shown that for the fixed database version of LZ78 the redundancy is *lower bounded* by $\Omega(\log\log n/\log n)$.

Although these results have great theoretical value, they are not completely satisfying since the same string s can be generated by ergodic sources with different entropies. In addition, most of these results do not hold for every string but only in the average case or for a set of strings which has probability one in the underlying probabilistic model.

In order to get results which hold for every string, in this paper we analyze the performance of the Lempel–Ziv algorithms under a different perspective. Instead of making assumptions on the input, we consider the so-called *empirical entropy* which is based on a probability distribution defined implicitly by the input string. More precisely, for any string s, we define the kth order entropy $H_k(s)$ by looking at the number of occurrences of each symbol following each k-length substring inside s. We say that an algorithm is *coarsely optimal* if, for all k, there exists a function f_k, with $\lim_{n\to\infty} f_k(n) = 0$, such that for any string s the compression ratio $\rho(s)$ is bounded by

$$\rho(s) \le H_k(s) + f_k(|s|). \tag{1}$$

In other words, an algorithm is coarsely optimal if its compression ratio differs from $H_k(s)$ by a quantity which depends only on the length $|s|$ and vanishes as $|s| \to \infty$. The coarse optimality of LZ78 has been proven by Plotnik et al. [11, Corollary 3] and in section 4 we show that LZ77 is coarsely optimal as well.

Having defined optimality using (1), we can analyze an important phenomenon which went undetected using the previous approaches. If a string s is such that $\lim_{|s|\to\infty} H_k(s) = 0$, it is possible that (1) holds with a function f_k such that $H_k(s) = o(f_k(|s|))$. Hence, we could have a coarsely optimal algorithm with a compression ratio much higher than the entropy. To avoid this counterintuitive phenomenon, it is natural to require that the function f_k is such that $f_k(|s|) = o(H_k(s))$. Unfortunately, it is possible to prove that, even for $k = 0$, neither LZ78 nor LZ77 is optimal according to this more satisfactory definition (Lemmas 3.1 and 4.7). For this reason we introduce the concept of λ-optimality with respect to the kth order entropy H_k. Loosely speaking, the compression ratio of a λ-optimal algorithm must be bounded by $\lambda H_k(s) + o(H_k(s))$. Therefore, a λ-optimal algorithm is guaranteed to compress efficiently also the *low entropy strings*, that is, those strings such that $H_k(s) \to 0$.

In this paper we prove that LZ78 cannot be λ-optimal with respect to any H_k with $k \ge 0$. This is not surprising since it is well known that LZ78 is not able to compress efficiently long runs of identical symbols. Then, we describe an algorithm which combines LZ78 with RLE which is 3-optimal with respect to H_0. Finally, we prove that LZ77 is 8-optimal with respect to H_0, and that it cannot be λ-optimal with respect to H_k for any $k \ge 1$. Our techniques are of general interest since they rely on properties of LZ78 and LZ77 which are shared by other compression algorithms based on parsing such as LZW, LZC, and LZSS (see [1, 13] for a description of these variants).

In [14, 15] Savari analyzes several variants of LZ78 and LZ77 and compares their output size to the empirical entropy of the string. For the compression ratio of these algorithms Savari proves a bound of the form $g\big(k, |s|, H_k(s)\big) H_k(s) +$ lower order terms. The function g is given explicitly for each algorithm, and additional bounds are given for the case $H_k(s) = 0$. The analysis is not limited to the case in which each symbol depends on the k previous symbols, but considers also more general models. However, the results in [14, 15] are valid only if $H_k(s)/|s|$ is bounded away from zero and therefore do not apply to the low entropy strings. For this reason these bounds complement the coarse optimality results for LZ78 and LZ77, but are not directly related with the concept of λ-optimality, which is mainly concerned with the compression of the low entropy strings.

The results of this paper do not aim to substitute results based on probabilistic assumptions. The latter are very deep and rich and they provide information also for the nonprobabilistic setting (see, for example, the results on the empirical entropy in [11, 14, 15]). One of the merits of our worst case analysis is that, as we will see, it provides new insight on Lempel–Ziv algorithms by revealing strengths and weaknesses which cannot be properly analyzed in the probabilistic setting.

2. Definitions and notation.

Let s be a string of length n over the alphabet $\mathcal{A} = \{\alpha_1, \dots, \alpha_h\}$, and let n_i denote the number of occurrences of the symbol α_i inside s. We define the zeroth order entropy as

$$H_0(s) = -\sum_{i=1}^{h} \frac{n_i}{n} \log\left(\frac{n_i}{n}\right),$$

where we assume $0 \log 0 = 0$. The value $|s|H_0(s)$ represents the output size of an ideal compressor which uses $-\log \frac{n_i}{n}$ bits for coding the symbol α_i. Although, as we will see, it is by no means easy to compress a string up to its zeroth order entropy, often even this is not enough. For example, suppose we want to compress an English text. Clearly, we would like to take advantage not only of the different frequencies of the single letters, but also of the fact that these frequencies depend upon the context. The degree of compression that we can achieve by considering a context of length k, is expressed by the conditional kth order entropy $H_k(s)$ defined as follows. For any string w and $\alpha_i \in \mathcal{A}$, let $n_{w\alpha_i}$ denote the number of occurrences in s of the string w followed by α_i. Let $n_w = \sum_i n_{w\alpha_i}$. We define

$$H_k(s) = -\frac{1}{|s|} \sum_{w \in \mathcal{A}^k} \left(\sum_{i=1}^{h} n_{w\alpha_i} \log\left(\frac{n_{w\alpha_i}}{n_w}\right)\right). \tag{2}$$

The value $|s|H_k(s)$ is the output size of an ideal algorithm which uses $-\log(n_{w\alpha_i}/n_w)$ bits for the symbol α_i when it appears after the "context" w. This means that the code for α_i depends on the k characters preceding it. Note that we are assuming that this algorithm codes the first k characters of s for free. As k increases, the entropy $H_k(s)$ is defined in terms of longer, and therefore more accurate, contexts. It is not difficult to prove that H_k is a decreasing function of k.

Formula (2) is not the only possible definition of entropy in terms of frequencies of characters. Another definition found in the literature is

$$H'_k(s) = -\frac{1}{k} \sum_{w \in \mathcal{A}^k} \frac{n_w}{n} \log\left(\frac{n_w}{n}\right),$$

that is, H'_k is the first order entropy of the k-letter words. Disregarding low order terms, we have $H'_k(s) \simeq \frac{1}{k}[H_1(s) + \cdots + H_k(s)]$ (we get an equality if we consider also the k-letter words which wrap around the string s). Since $H_k(s)$ decreases with k, we have $H'_k(s) \geq H_k(s)$. Since we are interested in bounding the output size of compression algorithms in terms of the entropy, by considering H_k we get stronger results.

Having defined the kth order entropy, we can define the class of optimal compression algorithms. The natural extension of the definition given in the information theoretic setting (see, for example, [5]) is the following.

DEFINITION 2.1. *A compression algorithm* $\mathtt{A}$ *is* coarsely optimal*, if for all* $k \geq 0$ *there exists a function* f_k*, with* $\lim_{n\to\infty} f_k(n) = 0$*, such that for all strings* s *we have*

$$\frac{\mathtt{A}(s)}{|s|} \leq H_k(s) + f_k(|s|), \tag{3}$$

where $\mathtt{A}(s)$ *denotes the output size of algorithm* $\mathtt{A}$ *on input* s*.*

As we have already pointed out, the above definition is not completely satisfactory since, if $H_k(s) \ll f_k(|s|)$, we can have an optimal algorithm for which the compression ratio is much greater than the entropy. For this reason, we introduce a more restrictive definition of optimality in which we require that the difference between the compression ratio and the kth-order entropy is a lower order term.

DEFINITION 2.2. *A compression algorithm* $\mathtt{A}$ *is* λ-optimal *with respect to* H_k*, if it is coarsely optimal and there exists a function* g_k*, with* $g_k(t) = o(t)$*, such that for any string* s *with* $H_k(s) \neq 0$

$$\mathtt{A}(s) \leq \lambda |s| H_k(s) + g_k(\mathtt{A}(s)), \tag{4}$$

where $\mathtt{A}(s)$ *denotes the output size of algorithm* $\mathtt{A}$ *on input* s*.*

Note that (4) implies that $g_k(\mathtt{A}(s)) = o(|s|H_k(s))$. We use (4) since it is easier to prove that a certain quantity is a lower order term with respect to the output size of the algorithm rather than with respect to the usually unknown kth order entropy. Note also that λ-optimality is defined with respect to a single entropy H_k. As we will see, to ask (4) to hold for every $k \geq 0$ would be too strong a requirement. However, a λ-optimal algorithm must be coarsely optimal; hence (3) must hold for every $k \geq 0$. By studying the λ-optimality of an algorithm we measure how well it behaves when the input is a low entropy string; this is something which is not possible using the concept of coarse optimality alone.

In this paper we make use of the following lemma which is a generalization of a similar result proven in [5] for stationary ergodic sequences.

LEMMA 2.3. *Let* $y_1, \ldots, y_t$ *denote a parsing of the string* s *such that each word* y_i *appears at most* M *times. For any* $k \geq 0$ *we have*

$$t \log t \leq |s| H_k(s) + t \log\left(\frac{|s|}{t}\right) + t \log M + \Theta(kt + t)\,. \qquad \square \tag{5}$$

The proof of this lemma is given in the appendix. Note that if the words in the parsing are all distinct the above inequality becomes

$$t \log t \leq |s| H_k(s) + t \log\left(\frac{|s|}{t}\right) + \Theta(kt + t)\,. \tag{6}$$

In the rest of the paper we use the following notation. The length of the output produced by the compression algorithm $\mathtt{A}$ on input s is denoted by $\mathtt{A}(s)$. The compression ratio achieved is $\rho(s) = \mathtt{A}(s)/|s|$. Given a nonempty string s, s^- will denote the string obtained from s by removing the last character. If $|s| > 1$, we set $s^{--} = (s^-)^-$. Given k words $w_1, w_2, \ldots, w_k$, their concatenation will be denoted by $w_1 w_2 \cdots w_k$.

3. Compression of low entropy strings with LZ78.

In this section we consider the algorithm LZ78. First, we show that for certain strings LZ78 compression ratio is well above the zeroth order entropy H_0. Then we prove that by combining LZ78 with RLE we get a new algorithm which is 3-optimal with respect to H_0.

The algorithm LZ78 works as follows (see [1] for details). Assuming the words $w_1, w_2, \ldots, w_{i-1}$ have been already parsed, LZ78 selects the ith word as the longest word that can be obtained by adding a single character to a previous word. Hence, $w_i^- = w_j$ for some $j < i$, but unless w_i is the last word in the parsing, we have $w_i \neq w_k$, for all $k < i$. As soon as w_i has been found, LZ78 outputs an encoding of the pair (j, α_i), where j is such that $w_i^- = w_j$ and α_i is the last character of w_i. If we are working with an alphabet with h symbols, such encoding takes $\lceil \log i \rceil + \lceil \log h \rceil$ bits (all logarithms in this paper are taken to the base 2). Hence, if the input string is parsed into t words, the output of LZ78 has size $t \log t + t \log h + O(t)$. In the following we assume that all the words are distinct since this greatly simplifies our analysis and changes the word count by at most one.

LEMMA 3.1. *There exists a constant $c > 0$ such that for every $n \geq 1$ there exists a string s of length n satisfying* $\mathtt{LZ78}(s)/|s| \geq c\sqrt{n}H_0(s)$.

Proof. Consider the string $s = 01^{(n-1)}$. LZ78 parses s into $\Theta(\sqrt{n})$ words, hence the compression ratio is $\Theta((\log n)/\sqrt{n})$, whereas $H_0(s) = (\log n)/n$. □

Despite all the optimality results for LZ78, we can hardly say that it compresses optimally the string s. As we have already pointed out, using the standard definition of optimal compression given by (3), the problem is that if $H_k(s) \to 0$ and $H_k(s) = o(f_k(|s|))$, the compression ratio $\rho(s)$ can be much higher than $H_k(s)$.

Lemma 3.1 suggests that the inability of LZ78 to compress efficiently low entropy strings is due to the inability to cope with long runs of identical symbols. In view of this, it is natural to ask if we can compress optimally also the low entropy strings by combining LZ78 with RLE. RLE is a technique frequently used when the input may contain long runs of identical symbols. RLE is used either as a fast stand-alone compressor, or as a preprocessing step for more complex compression procedures. We now present an algorithm, which we call $\mathtt{LZ78}_{RL}$, which follows the latter approach and uses RLE as a preprocessing step for LZ78.

3.1. The algorithm $\mathtt{LZ78}_{RL}$.

In this section we describe and analyze an algorithm which combines LZ78 with RLE. We underline that our analysis does not rely on the particular parsing realized by LZ78. Indeed, the analysis is based only on the two facts that LZ78 parses the input string into distinct words, and that its output size is $t \log t + O(t)$, t being the number of words in the parsing. Hence, our analysis can be applied also to other parsing-based compression algorithms. With some additional work it can be applied also to the algorithms LZW and LZC (the core of the Unix utility compress) in which the same word can appear in the parsing more than once.

Let s denote a string over the alphabet $\mathcal{A}$, and let s' be a substring of s. We call s' an *α-segment* if it contains only the symbol α, and the two symbols adjacent to s' are different from α. Let $\mathtt{0}$ denote a symbol not belonging to $\mathcal{A}$. For $m > 0$, $\alpha \in \mathcal{A}$, let $B(\alpha^m)$ denote the string obtained by writing m in binary using α as 1 and $\mathtt{0}$ as 0. For example, $B(\alpha\alpha\alpha\alpha\alpha) = \alpha\mathtt{0}\alpha$. Given the string s, we define the run length encoding of s as the string $\tilde{s}$ obtained from s by replacing each δ-segment δ^m with the string $B(\delta^m)$. For example, if $s = \alpha\alpha\alpha\beta\beta\alpha\gamma\gamma\gamma\gamma$, then

$$\tilde{s} = \alpha\alpha\beta\mathtt{0}\alpha\gamma\mathtt{00}.$$

Clearly, given $\tilde{s}$ we can obtain s. In fact, δ is always the first symbol in $B(\delta^m)$, and we know that we have reached the end of a δ-segment when we encounter a symbol different from δ or $\mathtt{0}$. Note that, since $|B(\delta^m)| = \lfloor \log m \rfloor + 1 \le m$, we have $|\tilde{s}| \le |s|$. In the following $\mathtt{LZ78}_{RL}$ will denote the algorithm which applies $\mathtt{LZ78}$ to the string $\tilde{s}$.

Our implementation of RLE is certainly somewhat different from the more common strategy to use a byte to store a repetition count for the last seen character. However, this simpler strategy is not powerful enough to make LZ78 overcome the problem outlined in Lemma 3.1. An obvious drawback of our RLE implementation is that it requires an extra character not belonging to $\mathcal{A}$. Note that this is not a problem when RLE is used together with algorithms like LZW and LZC which never output single characters. We have also devised a RLE strategy in which we do not need to extend the alphabet. However, this alternative encoding is less efficient, and when combined with LZ78 it produces an algorithm which is 6-optimal with respect to H_0 ($\mathtt{LZ78}_{RL}$ is 3-optimal with respect to H_0; see Theorem 3.6).

THEOREM 3.2. *The algorithm* $\mathtt{LZ78}_{RL}$ *is coarsely optimal with respect to the entropy* $H_k(s)$*, for any* $k \ge 0$.

Proof. Let t denote the number of words in the LZ78 parsing of the string $\tilde{s}$. In the Appendix (Lemma B.1) we show that this parsing induces a t-word parsing of the string s in which each word appears at most $2\log|s|$ times. By (5) we get

$$t\log t \le |s|H_k(s) + t\log\left(\frac{|s|}{t}\right) + t\log\log|s| + \Theta(kt)\,. \tag{7}$$

Since LZ78 parses $\tilde{s}$ into distinct words, we have (see, for example, [5, Lemma 12.10.1]) $t = O\Big(\frac{|\tilde{s}|}{\log|\tilde{s}|}\Big) = O\Big(\frac{|s|}{\log|s|}\Big)$. Hence, $\log|s| = O\Big(\frac{|s|}{t}\Big)$, and (7) becomes

$$t\log t \le |s|H_k(s) + 2t\log\left(\frac{|s|}{t}\right) + \Theta(kt)\,.$$

Since $t = O\Big(\frac{|s|}{\log|s|}\Big)$, we get

$$\frac{t\log t}{|s|} \le H_k(s) + O\biggl(\frac{\log\log|s|}{\log|s|} + \frac{k}{\log|s|}\biggr),$$

which proves the coarse optimality of $\mathtt{LZ78}_{RL}$. □

We now show that, in addition to being coarsely optimal, the algorithm $\mathtt{LZ78}_{RL}$ compresses almost optimally (with respect to H_0) also the low entropy strings. In our analysis we will use the following notation. Given a string s over the alphabet $\mathcal{A} = \{\alpha_1, \ldots, \alpha_h\}$, for $i = 1, \ldots, h$, let n_i denote the number of occurrences of α_i in s. We assume that $n_1 = \max_i n_i$, we set $r = n_2 + n_3 + \cdots + n_h$, and $\tau = \frac{|s|}{r}$. Define

$$G(n_1, n_2, \ldots, n_h) = |s|H_0(s) = -\sum_{i=1}^{h} n_i \log\left(\frac{n_i}{|s|}\right). \tag{8}$$

By setting $F(x) = x\log x$, we get the following alternative representation of G:

$$G(n_1, \ldots, n_h) = F(n_1 + n_2 + \cdots + n_h) - [F(n_1) + F(n_2) + \cdots + F(n_h)]\,. \tag{9}$$

In the following, we will use the two forms $|s|H_0(s)$ and $G(n_1, \dots, n_h)$ interchangeably.

Let t denote the number of words into which $\mathtt{LZ78}$ parses the string s. By (6), we know that

$$\mathtt{LZ78}(s) \le |s|H_0(s) + t\log\left(\frac{|s|}{t}\right) + \Theta(t)\,. \tag{10}$$

Using elementary calculus one can easily prove that

$$t\log(|s|/t) \le t\log\tau + \max[G(n_1, \dots, n_h) - t\log t, t\log\log t + 2t],$$

which yields

$$\mathtt{LZ78}(s) \le |s|H_0(s) + t\log\tau + O(t\log\log t)\,. \tag{11}$$

The following lemma formalizes the intuitive notion that as $\tau = \frac{|s|}{r}$ increases, the value $|s|H_0(s)$ becomes much smaller than $|s|$.

LEMMA 3.3. *For any string s, we have $H_0(s) \le \frac{\log(\tau h e)}{\tau}$.*

Proof. Let $n = |s|$. Using elementary calculus we get

$$\begin{aligned} G(n_1, \dots, n_h) &= G(n_1, n - n_1) + G(n_2, \dots, n_h) \\ &\le G(n - n/\tau, n/\tau) + (n/\tau)\log h \\ &= (n/\tau)[F(\tau) - F(\tau - 1) + \log h] \\ &\le (n/\tau)[\log(\tau e) + \log h], \end{aligned}$$

where we have used that F is convex and $F'(t) = \log(et)$. □

The following two lemmas bound the entropy and the length of the "encoded" string $\tilde{s}$ in terms of the entropy of s.

LEMMA 3.4. *For any string s, we have $|\tilde{s}|H_0(\tilde{s}) \le |s|H_0(s) + |\tilde{s}|$.*

Proof. For $i = 1, \dots, h$, let m_i denote the number of occurrences of α_i inside $\tilde{s}$, and z_i denote the number of $\mathtt{0}$'s created converting the α_i-segments. Let $v_i = m_i + z_i$. By (9), we have

$$\begin{aligned} |\tilde{s}|H_0(\tilde{s}) &= G\left(m_1, \dots, m_h, \sum_{i=1}^{h} z_i\right) \\ &= F\left(m_1 + \cdots + m_h + \sum_{i=1}^{h} z_i\right) - \left[\sum_{i=1}^{h} F(m_i) + F\left(\sum_{i=1}^{h} z_i\right)\right] \\ &\le F(v_1 + \cdots + v_h) - \sum_{i=1}^{h}[F(m_i) + F(z_i)]. \end{aligned}$$

By Jensen inequality, we know that $F(m_i) + F(z_i) \ge 2F(v_i/2) = F(v_i) - v_i$, hence

$$\begin{aligned} |\tilde{s}|H_0(\tilde{s}) &\le F(v_1 + \cdots + v_h) - \sum_{i=1}^{h}[F(v_i) - v_i] \\ &= G(v_1, \dots, v_h) + \sum_{i=1}^{h} v_i \\ &\le |s|H_0(s) + |\tilde{s}|. \qquad \square \end{aligned}$$

LEMMA 3.5. *For any string s, we have $|\tilde{s}| \le 2|s|H_0(s)$.*

Proof. Let $n_1 = \max_i n_i$, $r = |s| - n_1 = n_2 + \cdots + n_h$. If $n_1 \le |s|/2$, we have

$$|s|H_0(s) = \sum_{i=1}^{h} n_i \log\left(\frac{|s|}{n_i}\right) \ge \sum_{i=1}^{h} n_i = |s| \ge |\tilde{s}|,$$

and the lemma follows. If $n_1 > |s|/2$, then

$$\begin{aligned}|s|H_0(s) &\ge n_1 \log\left(\frac{n_1 + r}{n_1}\right) + \sum_{i=2}^{h} n_i \log\left(\frac{|s|}{r}\right)\\ &= r \log\left(1 + \frac{r}{n_1}\right)^{\frac{n_1}{r}} + r\log\left(\frac{|s|}{r}\right).\end{aligned}$$

Since, $(1 + 1/t)^t \ge 2$, for $t \ge 1$, we have

$$|s|H_0(s) \ge r + r\log\left(\frac{|s|}{r}\right). \tag{12}$$

In order to bound $|\tilde{s}|$, we note that the string s contains at most $r + 1$ α_1-segments. Hence,

$$|\tilde{s}| \le \sum_{i=1}^{r+1} (\log q_i + 1) + \sum_{i=2}^{h} n_i,$$

with $q_1 + \cdots + q_{r+1} = n_1$. From the concavity of $\log t$ we obtain

$$|\tilde{s}| \le (r + 1)\log\left(\frac{n_1}{r + 1}\right) + 2r + 1.$$

For $r = 1$, we have

$$\frac{|\tilde{s}|}{2|s|H_0(s)} \le \frac{2\log n_1 + 1}{2 + 2\log(n_1 + 1)} \le 1.$$

For $r \ge 2$, we set $t = n_1/r$, and we get

$$\frac{|\tilde{s}|}{2|s|H_0(s)} \le \frac{2r + (r + 1)\log t + 1}{2r + 2r\log(1 + t)}.$$

A straightforward computation shows that for $t \ge 1$, $2r\log(1 + t) \ge (r + 1)\log t + 1$, and the lemma follows. □

We are now ready to establish the main result of this section.

THEOREM 3.6. *The algorithm $\mathtt{LZ78}_{RL}$ is 3-optimal with respect to H_0.*

Proof. We have already shown that $\mathtt{LZ78}_{RL}$ is coarsely optimal (Theorem 3.2). Hence, we need to prove that for any string s with $H_0(s) \ne 0$ we have

$$\mathtt{LZ78}_{RL}(s) \le 3|s|H_0(s) + \text{lower order terms}.$$

By (11) we know that

$$\mathtt{LZ78}_{RL}(s) \le |\tilde{s}|H_0(\tilde{s}) + t\log\tau + O(t\log\log t),$$

where t denotes the number of words in the LZ78 parsing of the string $\tilde{s}$. If $\tau \leq 16h$ the theorem trivially holds, since, by Lemmas 3.4 and 3.5, we have $|\tilde{s}|H_0(\tilde{s}) \leq 3|s|H_0(s)$. Assume now $\tau > 16h$. By (10), we know that

$$\mathtt{LZ78}_{RL}(s) \leq |\tilde{s}|H_0(\tilde{s}) + t\log\left(\frac{|\tilde{s}|}{t}\right) + O(t)\,.$$

It is straightforward to verify that, for $0 < x < |\tilde{s}|$, we have $x\log(\tilde{s}/x) \leq (|\tilde{s}|\log e)/e \leq (2|\tilde{s}|)/3$. Hence, we can write

$$\mathtt{LZ78}_{RL}(s) \leq |\tilde{s}|H_0(\tilde{s}) + (2|\tilde{s}|)/3 + O(t)\,.$$

Since $\tau > 16h$, we have $\frac{\log(\tau he)}{\tau} \leq \frac{5}{6}$. Thus, by Lemmas 3.3 and 3.5 we have

$$\begin{aligned}\mathtt{LZ78}_{RL}(s) &\leq |\tilde{s}|\left(\frac{\log(\tau he)}{\tau}\right) + \frac{2}{3}|\tilde{s}| + O(t)\\ &\leq \frac{3}{2}|\tilde{s}| + O(t)\\ &\leq 3|s|H_0(s) + O(t)\,.\end{aligned}$$

This completes the proof. □

4. Compression of low entropy strings with LZ77.

In this section we consider the algorithm LZ77. First, we prove that this algorithm is coarsely optimal according to Definition 2.1. This is not surprising since it is known that LZ77 compresses much better than LZ78. In view of the results of section 3, what is more interesting is to understand how well LZ77 compresses the low entropy strings. To this end, we prove that LZ77 is 8-optimal with respect to H_0. We also show that this result cannot be substantially improved: we present a family of low entropy strings for which $\mathtt{LZ77}(s) \geq 2.5|s|H_0(s)$ and we prove that LZ77 cannot be λ-optimal with respect to H_1.

The algorithm LZ77 works as follows (see [1] for details). Assuming the words w_1, $w_2, \ldots, w_{i-1}$ have been already parsed, LZ77 selects the ith word as the longest word that can be obtained by adding a single character to a substring of $(w_1w_2\cdots w_i)^{--}$. Hence, w_i has the property that w_i^- is a substring of $(w_1\cdots w_i)^{--}$, but w_i is not a substring of $(w_1\cdots w_i)^-$. Note that although this is a recursive definition there is no ambiguity. In fact, if $|w_i| > 1$ at least the first character of w_i belongs to $w_1w_2\cdots w_{i-1}$. Once w_i has been found, LZ77 outputs an encoding of the triplet (p_i, l_i, α_i), where p_i is the starting position of w_i^- within $w_1w_2\cdots w_{i-1}$, $l_i = |w_i|$, and α_i is the last character of w_i.

Compared to LZ78, the algorithm LZ77 parses the input into fewer words and generally achieves a better compression. In addition, it still has the nice property that both coding and decoding can be done in $O(|s|)$ time (see [12]). Note that in the original formulation of LZ77 [23] p_i is allowed to denote a position only within the last L characters of $w_1w_2\cdots w_{i-1}$ (the so called “sliding window”). The use of a sliding window makes the coding procedure faster, and in most cases it affects the compression ratio only slightly. However, as we will see (Lemma 4.3), if the window has a fixed size the algorithm cannot be coarsely optimal.

4.1. Coarse optimality of LZ77.

To analyze the compression ratio achieved by LZ77 we need to specify the encoding of the triplet (p_i, l_i, α_i) representing the ith word in the parsing of s. For our analysis we assume the following simple scheme.

Since $1 \le p_i \le \sum_{j<i} l_j$, we use $\left\lceil \log(\sum_{j<i} l_j) \right\rceil$ bits for encoding p_i. Assuming the input alphabet has h symbols, we use $\lceil \log h \rceil$ bits for encoding α_i. Finally, since we cannot bound in advance the size of l_i, we use a scheme which allows the encoding of unbounded integers. More precisely, we code l_i using $1 + \lfloor \log l_i \rfloor + 2\lfloor \log(1 + \lfloor \log i \rfloor) \rfloor$ bits (for details on this and other similar schemes, see, for example, [3]). Although there are other possible methods for describing these triplets, we believe our analysis can be adapted to all "reasonable" encodings.

Assuming we use the above encoding scheme for each word in the parsing, the output of LZ77 has size

$$\texttt{LZ77}(s) = \sum_{i=1}^{t} \Big(\log(\sum\nolimits_{j<i} l_j) + \log l_i + 2\log(1 + \log l_i) \Big) + O(t)\,.$$

Obviously, $(\sum_{j<i} l_j) \le |s|$. In addition, from the concavity of the function $\log x + 2\log(1+\log x)$, we get $\sum_i (\log l_i + 2\log(1+\log l_i)) \le t[\log(|s|/t) + 2\log(1+\log(|s|/t))]$. Hence,

$$\texttt{LZ77}(s) \le t\log|s| + t\log\left(\frac{|s|}{t}\right) + O(t\log\log(|s|/t))\,,$$

or, equivalently,

$$\texttt{LZ77}(s) \le t\log t + 2t\log\left(\frac{|s|}{t}\right) + O(t\log\log(|s|/t))\,. \tag{13}$$

Since LZ77 parses its input into distinct words, we can use (6) which yields

$$\texttt{LZ77}(s) \le |s|H_k(s) + 3t\log\left(\frac{|s|}{t}\right) + O\left(kt + t\log\log\left(\frac{|s|}{t}\right)\right). \tag{14}$$

We can now easily prove that LZ77 is optimal according to Definition 2.1.

THEOREM 4.1. *The algorithm* LZ77 *is coarsely optimal.*

Proof. Let t denote the number of words in the LZ77 parsing. Since the words are distinct, we have $t = O\Big(\frac{|s|}{\log|s|}\Big)$. From (14) we get

$$\frac{\texttt{LZ77}(s)}{|s|} \le H_k(s) + O\left(\frac{\log\log|s|}{\log|s|} + \frac{k}{\log|s|}\right),$$

which proves the optimality of LZ77. □

Despite the above optimality result, the following lemma shows that, if $H_1(s) \to 0$, LZ77 compression ratio can be asymptotically greater than H_1.

LEMMA 4.2. *There exists a constant $c > 0$ such that for every $n > 1$ there exists a string s of length $|s| \ge n$ satisfying*

$$\frac{\texttt{LZ77}(s)}{|s|} \ge c\,\frac{\log|s|}{\log\log|s|} H_1(s).$$

Proof. Consider the following string:

$$s = 1\,0^k\,2^{2^k}\,1\,101\,10^2 1\,10^3 1\,10^4 1 \cdots 10^k 1.$$

We have $|s| = 2^k + O\big(k^2\big)$. A simple computation shows that $|s|H_1(s) = k \log k + O(k)$. The LZ77 parsing of s is

$$1 \quad 0 \quad 0^{(k-1)}2 \quad 2^{(2^k-1)}1 \quad 101 \quad 10^21 \quad 10^31 \quad \cdots \quad 10^k1.$$

For $i \geq 5$, we have $\sum_{j<i} l_j > 2^k$. Hence, the encoding of p_i takes $\Omega(k)$ bits. Since there are $k+4$ words, we have $\mathtt{LZ77}(s) = \Omega\big(k^2\big)$ and the lemma follows. □

Let us comment on the above lemma. We can clearly see that the inefficiency of LZ77 is due to the cost of encoding p_i, and it is possible that, using a clever encoding, we can reduce LZ77 output size significantly. However, for any encoding scheme we have been able to think of, it was always possible to find a "bad" string which LZ77 is not able to compress up to the first order entropy. Lemma 4.2 also shows that it is sometimes not convenient to search for the longest match over the entire parsed portion of the input string. Indeed, a common practice is to restrict the search to a sliding window containing the last L characters seen by the algorithm (this is in fact the original formulation of LZ77 found in [23]). This strategy, which we call $\mathtt{LZ77}_L$, usually reduces dramatically the cost of encoding the pointers p_i's, at the expense of a moderate increment in the number of words. For example, for $k < L < 2^k$, $\mathtt{LZ77}_L$ parses the string given in the proof of Lemma 4.2 in $k+5$ words. The output size is $\approx k(\log k + \log L)$ which, for $L = k^{O(1)}$, differs from $|s|H_1(s)$ only by a constant factor. Unfortunately, the following lemma shows that $\mathtt{LZ77}_L$ is not coarsely optimal since it sometimes fails to compress up to the kth order entropy when $k \geq L-1$.

LEMMA 4.3. *The algorithm* $\mathtt{LZ77}_L$ *is not coarsely optimal for any* $L > 0$.

Proof. For $k = L-1$ and $n > 0$, we exhibit a string s of length $2kn+1$ such that $|s|H_k(s) = \Theta(\log n)$ and $\mathtt{LZ77}_L(s) = \Theta(n)$. Let

$$s = (0^k1^k)^n1.$$

The $\mathtt{LZ77}_L$ parsing of s consists of the following $2n+1$ words

$$0 \quad 0^{k-1}1 \quad 1^{k-1}0 \quad 0^{k-1}1 \quad \cdots \quad 1^{k-1}0 \quad 0^{k-1}1 \quad 1^k.$$

To see this, consider for example the situation after the parsing of the third word. The unparsed portion of the string is $0^{k-1}1^k0^k1^k0^k\cdots$. Since at that moment the sliding window contains 1^k0, the only possible match is the one starting at the last character of the sliding window. Hence, every word has length at most k and $\mathtt{LZ77}_L(s) = \Theta(n)$. Note that LZ77 parses the above string into four words only.

For the computation of $H_k(s)$ we notice that s contains only $2k$ distinct k-letter words, namely

$$0^i1^{k-i}, \quad i = 1, \ldots, k \qquad \text{and} \qquad 1^i0^{k-i} \quad i = 1, \ldots, k.$$

In addition, every occurrence of a word 0^i1^{k-i} is always followed by a 1, and, for $i < k$, every occurrence of 1^i0^{k-i} is followed by a 0. Finally, the word 1^k is followed $n-1$ times by a 0 and once by a 1. As a result, we have $|s|H_k(s) = \Theta(\log n)$ and the lemma follows. □

4.2. $\boldsymbol{\lambda}$-optimality of LZ77.

We now show that LZ77 is 8-optimal with respect to H_0. We start our analysis with the following lemma which establishes an optimality result for the number of words in the LZ77 parsing.

LEMMA 4.4. *Let* $w_1, w_2, \ldots, w_t$ *denote the* LZ77 *parsing of the string* s. *Suppose* $y_1, y_2, \ldots, y_{t'}$ *is another parsing of* s *such that, for* $i = 1, \ldots, t'$, $|y_i| = 1$ *or* y_i^- *is a substring of* $(y_1 \cdots y_i)^{--}$. *Then,* $t \leq t'$.

Proof. One can easily prove by induction that, for $j = 1, \ldots, t$, the string $y_1 y_2 \cdots y_j$ is a prefix of $w_1 w_2 \cdots w_j$. □

Given a string s, in the following n_i will denote the number of occurrences of α_i in s. We assume $n_1 = \max_i n_i$, and we set $r = |s| - n_1$. The following lemma provides a bound on the number of words in the `LZ77` parsing in terms of r. The bound is tight when $r \ll |s|$.

LEMMA 4.5. *Let t denote the number of words in which the algorithm* `LZ77` *parses the string s. We have $t \leq 2(r+1)$.*

Proof. We prove the lemma by showing that there exists a parsing $y_1 y_2 \cdots y_{t'}$, $t' \leq 2(r+1)$, for s which satisfies the hypothesis of Lemma 4.4. Let k denote the number of α_1-segments appearing inside s. Since there are r characters different from α_1 we have $1 \leq k \leq r+1$. Consider the following parsing. Each α_1-segment and the character following it is parsed in two words: the first consisting of the single character α_1, the second of the form $\alpha_1^k \alpha_j$. After this, we are left with at most $r-k+1$ unparsed characters, all of them different from α_1. By parsing these characters using length-one words, we get a parsing for s consisting of $2k + (r-k+1) \leq 2(r+1)$ words. □

We are now ready to establish the main result of this section.

THEOREM 4.6. *The algorithm* `LZ77` *is* 8*-optimal with respect to H_0.*

Proof. We have already shown that `LZ77` is coarsely optimal (Theorem 4.1). Hence, we need to prove that, for any string s with $H_0(s) \neq 0$, we have

$$\texttt{LZ77}(s) \leq 8|s|H_0(s) + \text{lower order terms.}$$

Let t denote the number of words in the `LZ77` parsing of s, and let r be defined as in Lemma 4.5. We consider four cases.

Case 1. $1 \leq r < 6$.

By Lemma 4.5 we know that $t \leq 12$. Using (13) we get

$$\texttt{LZ77}(s) \leq 2t\left(\log\frac{|s|}{t}\right) + O(t \log\log(|s|/t)). \tag{15}$$

Let $g(x) = x\log(|s|/x)$. For $0 < x < (|s|/e)$, we have $g'(x) > 0$. Since $r+1 \leq 2r$, we get

$$t\log\left(\frac{|s|}{t}\right) \leq 4r\log\left(\frac{|s|}{r}\right) \leq 4\left[\sum_{i=2}^{h} n_i \log\left(\frac{|s|}{n_i}\right)\right] \leq 4|s|H_0(s).$$

Combining this inequality with (15) we get the thesis.

Case 2. $6 \leq r < 3|s|/7e$.

By Lemma 4.5 we know that $t \leq 2(r+1) \leq \frac{7}{3}r \leq |s|/e$. Reasoning as in Case 1 we get

$$t\log\left(\frac{|s|}{t}\right) \leq \frac{7}{3}\left[r\log\left(\frac{|s|}{r}\right)\right] \leq \frac{7}{3}|s|H_0(s).$$

Substituting this inequality into (14) we get the thesis.

Case 3. $3|s|/7e \leq r < |s|/2$.

Let $G(n_1, n_2)$ be defined as in (8). We have

$$|s|H_0(s) \geq G(|s|-r, r) = |s|G(1 - \tfrac{r}{|s|}, \tfrac{r}{|s|}) \geq |s|G\left(1 - 3/(7e), 3/(7e)\right) \geq |s|/2.$$

By elementary calculus we know that, for $0 < x < |s|$, $x \log(|s|/x) \leq (|s|/e) \log e \leq 2|s|/3$. Hence

$$t \log\left(\frac{|s|}{t}\right) \leq \frac{2|s|}{3} \leq \frac{4}{3}|s|H_0(s).$$

Substituting this inequality into (14) we get the thesis.

Case 4. $r \geq |s|/2$.

Since, for $i = 1, \ldots, h$, $n_i < |s|/2$ we have

$$|s|H_0(s) = \sum_{i=1}^{h} n_i \log\left(\frac{|s|}{n_i}\right) \geq |s|.$$

Combining this inequality with the fact that $t \log(|s|/t) \leq 2|s|/3$ we get $\mathtt{LZ77}(s) \leq 3|s|H_0(s) + O(t \log\log(\frac{|s|}{t}))$, and the theorem follows. □

Finally, we show that the above theorem cannot be substantially improved since there exists an infinite family of strings such that the compression ratio of LZ77 is greater than $2.5H_0$.

LEMMA 4.7. *For every n, there exists a string s of length $|s| \geq n$ such that* $\frac{\mathtt{LZ77}(s)}{|s|} \geq 2.5H_0(s)$.

Proof. Consider the following string

$$s = 010^4 10^9 1 \cdots 10^{i^2} 1 \cdots 10^{k^2} 1.$$

We have $|s| = k^3/3 + O(k^2)$, and $|s|H_0(s) = 2k \log k + O(k)$. The LZ77 parsing of s is

$$0 \quad 1 \quad 00 \quad 0^2 10^5 \quad 0^4 10^{10} \quad \cdots \quad 0^{2(i-1)} 10^{i^2+1} \quad \cdots \quad 0^{2(k-2)} 10^{(k-1)^2+1} \quad 0^{2(k-1)} 1.$$

Let p_i and l_i denote, respectively, the starting position and the length of the ith word. We have $l_i = \Theta(i^2)$. Hence, neglecting the $\log\log$ term, encoding l_i takes $\approx 2 \log i + \log\log i$ bits. Similarly, since $\sum_{j<i} l_j = \Theta(i^3)$, encoding p_i takes $\approx 3 \log i$ bits. Neglecting lower order terms, the output size of LZ77 is therefore $5(\sum_{i \leq k} \log i) \approx 5k \log k$ bits, and the lemma follows. □

5. Conclusions.

In order to analyze the performance of the Lempel–Ziv algorithms without any assumption on the input, we have compared the compression ratio of LZ77 and LZ78 with the so-called empirical entropy of the input string. We have shown that the standard definition of optimal compression does not take into account the performance of compression algorithms when the input is a low entropy string. For this reason we have introduced the concept of λ-optimality which makes it possible to measure how well an algorithm performs when the input is a low entropy string. We have proven that by combining LZ78 with RLE we get an algorithm which is 3-optimal with respect to H_0, and that LZ77 is 8-optimal with respect to H_0.

A natural open question is whether there exist parsing-based compression algorithms which are λ-optimal with respect to H_k for $k \geq 1$. Theorem 4.2 shows that LZ77 is not λ-optimal for $k \geq 1$ and one can easily verify that the same is true for $\mathtt{LZ78}_{RL}$ as well. Theorem 4.2 also shows that to improve LZ77 performance one should try to reduce the cost of the backward pointers. Some interesting ideas in this direction are described in [1, section 3.4] and [4]. Lemma 4.3 shows that the simple use of a fixed size sliding window does not yield an optimal algorithm. However, the algorithm $\mathtt{LZ77}_L$ deserves further investigation for several reasons. First, it is the variant which is the basis for the most popular compressors (zip, gzip, arj, lha, zoo, etc.). Second, we have been able to show that its compression ratio can be higher than H_k only when $k \geq L-1$. Since in typical implementations $L \approx 10^5$, our result has only a theoretical value. An interesting open problem is to prove or disprove that for $\texttt{LZ77}_L$ inequalities (3) and (4) hold for $k \leq \theta(L)$ for some appropriate function θ.

Appendix A. Proof of Lemma 2.3. In this appendix we prove Lemma 2.3. Our proof follows closely the proof given in [5, section 12.10] for a similar result involving the entropy of a stationary ergodic source.

Let $s = x_1 x_2 \cdots x_{n+k}$ be a string of length $n+k$. For $w \in \mathcal{A}^k$ and $\alpha \in \mathcal{A}$ let $n_{w\alpha}$ and n_w be defined as in section 2. We define the empirical probability $P(w\alpha)$ as $P(w\alpha) = n_{w\alpha}/n_w$ (if $n_w = 0$ we set $P(w\alpha) = 0$). For $i > k$ let s_i denote the length-k string preceding x_i in s. Using this notation, the kth order entropy can be written as

$$|s|H_k(s) = -\sum_{i=k+1}^{n+k} \log P(s_i x_i). \tag{16}$$

Let $v = v_1 v_2 \cdots v_h$ and $w = w_1 \cdots w_k$ be two strings over $\mathcal{A}$. For $i = 1, \ldots, h$ let $w^{(i)}$ denote the length-k string preceding the symbol v_i in wv (for example, $w^{(2)} = w_2 \cdots w_k v_1$). We define

$$\mathcal{P}(wv) = \prod_{i=1}^{h} P(w^{(i)} v_i). \tag{17}$$

The value $\mathcal{P}(wv)$ corresponds to the conditional probability $P(v|w)$ introduced in Lemma 12.10.3 of [5]. The following lemma shows that, in some sense, $\mathcal{P}$ does behave as a conditional probability.

Lemma A.1. *For any string w and $l \geq 1$ we have $\sum_{v \in \mathcal{A}^l} \mathcal{P}(wv) \leq 1$.*

Proof. We prove the result by induction on l. If $l = 1$ we have $\sum_{\alpha \in \mathcal{A}} \mathcal{P}(w\alpha) = \sum_{\alpha \in \mathcal{A}} P(w\alpha) \leq 1$. For $l > 1$, let $v = v_1 \cdots v_l$ and $v' = v_2 \cdots v_l$. We have

$$\begin{aligned} \sum_{v \in \mathcal{A}^l} \mathcal{P}(wv) &= \sum_{v \in \mathcal{A}^l} P(wv_1) \mathcal{P}(w^{(2)} v') \\ &= \sum_{\alpha \in \mathcal{A}} P(w\alpha) \left(\sum_{v' \in \mathcal{A}^{l-1}} \mathcal{P}(w^{(2)} v') \right) \\ &\leq \sum_{\alpha \in \mathcal{A}} P(w\alpha) \\ &\leq 1. \qquad \square \end{aligned}$$

Corollary A.2. *Let $V = \{v_1, \ldots, v_m\}$ be a collection of length-l strings in which each v_i appears at most M times. For any string w we have*

$$\sum_{v \in V} \mathcal{P}(wv) \leq M. \qquad \square$$

Let $y_1, \ldots, y_c$ denote any parsing of the string $x_{k+1} \cdots x_{n+k}$, and let z_i denote the length-k substring preceding y_i in $s = x_1 \cdots x_{n+k}$. For $l \geq 1$ and $w \in \mathcal{A}^k$ we define c_{lw} as the number of words y_i such that $|y_i| = l$ and $z_i = w$. By construction, the values c_{lw} satisfy

$$\sum_{l,w} c_{lw} = c, \qquad \sum_{l,w} l c_{lw} = n. \tag{18}$$

The following lemma is a generalization of Ziv's inequality [5, Lemma 12.10.3].

LEMMA A.3. *Let $y_1, \ldots, y_c$ denote a parsing of the string $x_{k+1} \cdots x_{n+k}$ in which each word appears at most M times. We have*

$$|s|H_k(s) \geq \sum_{w \in \mathcal{A}^k,\, l \geq 1} c_{lw} \log c_{lw} - c \log M.$$

Proof. By (16) and (17) we have

$$\begin{aligned}
|s|H_k(s) &= -\sum_{i=k+1}^{n+k} \log P(s_i x_i) \\
&= -\sum_{i=1}^{c} \log \mathcal{P}(z_i y_i) \\
&= -\sum_{l,w} \sum_{|y_i|=l,\ z_i=w} \log \mathcal{P}(z_i y_i) \\
&= -\sum_{l,w} c_{lw} \left(\sum_{|y_i|=l,\ z_i=w} \frac{1}{c_{lw}} \log \mathcal{P}(z_i y_i) \right).
\end{aligned}$$

By Jensen's inequality and Corollary A.2 we get

$$\begin{aligned}
|s|H_k(s) &\geq -\sum_{l,w} c_{lw} \log\left(\sum_{|y_i|=l,\ z_i=w} \frac{1}{c_{lw}} \mathcal{P}(z_i y_i) \right) \\
&\geq -\sum_{l,w} c_{lw} \left(\log \frac{1}{c_{lw}} + \log\left(\sum_{|y_i|=l,\ z_i=w} \mathcal{P}(z_i y_i) \right) \right) \\
&\geq \sum_{l,w} c_{lw} \log c_{lw} - c \log M. \qquad \square
\end{aligned}$$

THEOREM A.4. *Let $y_1, \ldots, y_c$ denote a parsing of the string $x_{k+1} \cdots x_{n+k}$ in which each word appears at most M times. We have*

$$c \log c \leq |s|H_k(s) + c \log M + c \left(k \log |\mathcal{A}| + \log \frac{n}{c} \right) + \Theta(c)\,.$$

Proof. Let $\pi_{lw} = c_{lw}/c$. We have

$$\begin{aligned}
\sum_{l,w} c_{lw} \log c_{lw} &= c \left(\sum_{l,w} \frac{c_{lw}}{c} (\log \frac{c_{lw}}{c} + \log c) \right) \\
&= c \log c + c \left(\sum_{l,w} \pi_{lw} \log \pi_{lw} \right).
\end{aligned}$$

By (18), the values π_{lw} satisfy

$$\sum_{l,w} \pi_{lw} = 1, \qquad \sum_{l,w} l \pi_{lw} = n/c. \tag{19}$$

Using Lagrange multipliers to maximize $-\sum_{l,w} \pi_{lw} \log \pi_{lw}$ under the constraint (19), or reasoning as in the proof of [5, Theorem 12.10.1] we get

$$-\sum_{l,w} \pi_{lw} \log \pi_{lw} \leq k \log |\mathcal{A}| + \log \left(1 + \frac{n}{c} \right) + \frac{n}{c} \log \left(1 + \frac{c}{n} \right).$$

Combining the above inequalities with Lemma A.3 we get

$$c\log c \le |s|H_k(s) + c\log M + c\left(k\log|\mathcal{A}| + \log\left(1+\frac{n}{c}\right) + \frac{n}{c}\log\left(1+\frac{c}{n}\right)\right).$$

The thesis follows observing that $\log(1+c/n)^{(n/c)} \le \log e$ and $\log(1+n/c) \le \log(n/c) + (c/n)\log e$. □

Proof of Lemma 2.3. Let $s = x_1 \cdots x_n$ with $n > k$. Let $y_1, \ldots, y_t$ denote a parsing of s in which each word appears at most M times. This parsing induces a parsing of $x_{k+1} \cdots x_n$ in which each word appears at most $M+1$ times. The thesis follows by Theorem A.4. □

Appendix B. $\texttt{LZ78}_{RL}$-induced parsing.

LEMMA B.1. *Let $\tilde{s}$ be derived from s as described in section* 3, *and let $w_1, w_2, \ldots, w_t$ denote the* LZ78 *parsing of $\tilde{s}$. Then, it is possible to build a parsing $w'_1, w'_2, \ldots, w'_t$ of s such that each word appears at most* $2\log|s|$ *times.*

Proof. The parsing of $w'_1, w'_2, \ldots, w'_t$ is defined as follows. For $i = 1, \ldots, |\tilde{s}|$, we associate to the ith character of $\tilde{s}$ a nonempty string ω_i with the property that

$$s = \omega_1\omega_2\cdots\omega_{|\tilde{s}|}. \tag{20}$$

Then, from each word w_j in the parsing of $\tilde{s}$ we get the word w'_j by simply concatenating the strings ω_i's corresponding to the characters of w_j.

The strings ω_i's are defined by partially reversing the binary encoding utilized for the construction of the string $\tilde{s}$. More precisely, let $b = b_l b_{l-1} \cdots b_1 b_0$, $b_i \in \{\alpha, \mathtt{0}\}$ denote the encoding of the α-segment α^m (that is, with the notation of section 3 $b_l \cdots b_0 = B(\alpha^m)$). We associate to each character b_j the string $\omega(b_j)$ defined by

$$\omega(b_j) = \begin{cases} \alpha & \text{if } j = l;\\ \alpha^{2^j} & \text{if } j < l \text{ and } b_j = \mathtt{0};\\ \alpha^{2^{j+1}} & \text{if } j < l \text{ and } b_j = \alpha. \end{cases} \tag{21}$$

Note that, for $j < l$, $\omega(b_j)$ contains 2^j more α's than if we simply had reversed the binary encoding. This is done at the expense of b_l (which translates to a single α) with the purpose of ensuring that every $\omega(b_j)$ is nonempty. One can easily verify that $\omega(b_l)\omega(b_{l-1})\cdots\omega(b_0) = \alpha^m$, which proves that the strings ω_i's satisfy (20).

We now show that in the parsing $w'_1, w'_2, \ldots, w'_t$ each word appears at most $2\log|s|$ times. Since the words w_j's are distinct, we need to show that when we replace the single characters with the strings ω_i's, at most $2\log|s|$ distinct words translate into the same word. To prove this we introduce the following notation. For each substring σ of $\tilde{s}$ we denote by $\omega(\sigma)$ the string obtained by replacing each character of σ with the corresponding ω_i's. For $\alpha \in \mathcal{A}$, we denote with $B_{ij}(\alpha^m)$ the string obtained by removing from $B(\alpha^m)$ the first i and the last j characters. For example,

$$B_{10}(\alpha^5) = \mathtt{0}\alpha, \qquad B_{01}(\alpha^4) = \alpha\mathtt{0}, \qquad B_{00}(\alpha^{17}) = \alpha\mathtt{000}\alpha.$$

Finally, given any string $\sigma \in \{\alpha, \mathtt{0}\}^*$ we denote with $\mathtt{Bin}(\sigma)$ the integer we get by replacing α with 1 and interpreting the result as a binary number. For example,

$$\mathtt{Bin}(\alpha) = \mathtt{Bin}(\mathtt{00}\alpha) = 1, \qquad \mathtt{Bin}(\alpha\mathtt{00}) = 4.$$

By construction we know that $\tilde{s}$ consists in a concatenation of "compressed" α-segments. Hence, a generic word w_i in the parsing of $\tilde{s}$ has the form[1]

$$w_i = B_{r0}(\alpha_{i_1}^{n_1})B(\alpha_{i_2}^{n_2}) \cdots B(\alpha_{i_{k-1}}^{n_{k-1}})B_{0s}(\alpha_{i_k}^{n_k}).$$

We have

$$\begin{aligned} \omega(w_i) &= \omega(B_{r0}(\alpha_{i_1}^{n_1}))\, \omega(B(\alpha_{i_2}^{n_2}))\, \cdots\, \omega(B(\alpha_{i_{k-1}}^{n_{k-1}}))\, \omega(B_{0s}(\alpha_{i_k}^{n_k})) \\ &= \omega(B_{r0}(\alpha_{i_1}^{n_1}))\, \alpha_{i_2}^{n_2}\, \cdots\, \alpha_{i_{k-1}}^{n_{k-1}}\, \omega(B_{0s}(\alpha_{i_k}^{n_k})). \end{aligned} \tag{22}$$

Thus, in order to count how many distinct w_i's translate into the same word, we need to study when two distinct strings σ_1, σ_2 of the form

$$\sigma_1 = B_{i0}(\alpha^{n_1}) \qquad \sigma_2 = B_{j0}(\alpha^{n_2})$$

or

$$\sigma_1 = B_{0i}(\alpha^{m_1}) \qquad \sigma_2 = B_{0j}(\alpha^{m_2}),$$

are such that $\omega(\sigma_1) = \omega(\sigma_2)$.

Consider first the case $\sigma_1 = B_{i0}(\alpha^{n_1}), \sigma_2 = B_{j0}(\alpha^{n_2})$. By (21) we know that the number of α's in $\omega(\sigma_1)$ is given by

$$|\omega(\sigma_1)| = |\omega(B_{i0}(\alpha^{n_1}))| = \begin{cases} \mathtt{Bin}(\sigma_1), & \text{if } i = 0; \\ \mathtt{Bin}(\sigma_1) + 2^{|\sigma_1|} - 1, & \text{if } i > 0; \end{cases}$$

and a similar result holds for $\omega(\sigma_2)$. We now show that $\omega(\sigma_1) = \omega(\sigma_2)$ with $\sigma_1 \neq \sigma_2$ only if $i = 0$ and $j \neq 0$ or, vice versa, $i \neq 0$ and $j = 0$. If $i = j = 0$, then

$$|\omega(\sigma_1)| = |\omega(\sigma_2)| \quad \Rightarrow \quad \mathtt{Bin}(\sigma_1) = \mathtt{Bin}(\sigma_2) \quad \Rightarrow \quad \sigma_1 = \sigma_2.$$

In fact, we cannot have, say, $\sigma_1 = \mathtt{000}\sigma_2$, since $\sigma_1 = B_{00}(\alpha^{n_1})$ and its leading character is different from `0` by construction. Assume now $i, j \neq 0$. If $|\sigma_1| = |\sigma_2|$, then

$$|\omega(\sigma_1)| = |\omega(\sigma_2)| \quad \Rightarrow \quad \mathtt{Bin}(\sigma_1) = \mathtt{Bin}(\sigma_2),$$

which again implies $\sigma_1 = \sigma_2$ since the two strings have the same length. Finally, if $i, j \neq 0$ and $|\sigma_1| \neq |\sigma_2|$, for example, $|\sigma_1| > |\sigma_2|$, we have

$$\begin{aligned} |\omega(\sigma_1)| &= \mathtt{Bin}(\sigma_1) + 2^{|\sigma_1|} - 1 \\ &\geq 2^{|\sigma_2|} + 2^{|\sigma_2|} - 1 \\ &> \mathtt{Bin}(\sigma_2) + 2^{|\sigma_2|} - 1 \\ &= |\omega(\sigma_2)|, \end{aligned}$$

which implies $\omega(\sigma_1) \neq \omega(\sigma_2)$. In summary, we can conclude that there exist at most two distinct strings $\sigma_1 = B_{i0}(\alpha^{n_1}), \sigma_2 = B_{j0}(\alpha^{n_2})$ such that $\omega(\sigma_1) = \omega(\sigma_2)$.

Consider now the case $\sigma_1 = B_{0i}(\alpha^{m_1}), \sigma_2 = B_{0j}(\alpha^{m_2})$. By (21) we get that the number of α's in $\omega(\sigma_1)$ is given by

$$|\omega(\sigma_1)| = |\omega(B_{0i}(\alpha^{m_1}))| = \mathtt{Bin}(\sigma_1)2^i - 2^i - 1 = 1 + (\mathtt{Bin}(\sigma_1) - 1)2^i. \tag{23}$$

[1] We are assuming that w_i contains at least two distinct characters of $\mathcal{A}$. Therefore, we do not consider the case $w_i = B_{jk}(\alpha^n)$ which, however, can be handled with a similar analysis.

We observe that we can have $\omega(\sigma_1) = \omega(\sigma_2)$ with $\sigma_1 \neq \sigma_2$ only if $i \neq j$. In fact, if $i = j$, by (23) we have

$$|\omega(\sigma_1)| = |\omega(\sigma_2)| \quad \Rightarrow \quad \mathtt{Bin}(\sigma_1) = \mathtt{Bin}(\sigma_2),$$

which implies $\sigma_1 = \sigma_2$, since the leading character of both strings is different from 0. Note that, by construction, the length of every "compressed" α-segment $B(\alpha^n)$ is at most $\log|s|$. Hence, the above observation implies that there can be at most $\log|s|$ distinct strings $\sigma_1, \dots, \sigma_k$, $\sigma_i = B_{0j_i}(\alpha^{n_i})$, such that $\omega(\sigma_1) = \omega(\sigma_2) = \cdots = \omega(\sigma_k)$. In fact, the indices j_i's must be distinct and none of them can be greater than $\log|s|$.

We can now conclude our analysis of the induced parsing $w'_1, \dots, w'_t$. Since there are only two possible distinct prefixes and $\log|s|$ possible distinct suffixes which translate to the same substring, by (22) we have that at most $2\log|s|$ (distinct) words, w_i's, can translate to the same w'_j.

This completes the proof. □

REFERENCES

[1] T. Bell, I. Witten, and J. Cleary, *Modelling for text compression*, ACM Computing Surveys, 21 (1989), pp. 557–592.

[2] P. Bender and J. Wolf, *New asymptotic bounds and improvements on the Lempel-Ziv data compression algorithm*, IEEE Trans. Inform. Theory, 37 (1991), pp. 721–729.

[3] J. Bentley, D. Sleator, R. Tarjan, and V. Wei, *A locally adaptive data compression scheme*, Comm. ACM, 29 (1986), pp. 320–330.

[4] C. Bloom, *LZP: A new data compression algorithm*, in Proceedings of the IEEE Data Compression Conference, Snowbird, UT, 1996.

[5] T. M. Cover and J. A. Thomas, *Elements of Information Theory*, John Wiley & Sons, New York, 1991.

[6] G. Hansel, D. Perrin, and I. Simon, *Compression and entropy*, in Proceedings of the 9th Annual Symposium on Theoretical Aspects of Computer Science, Lecture Notes in Comput. Sci. 577, Springer, Berlin, 1992, pp. 515–528.

[7] P. Jacquet and W. Szpankowski, *Asymptotic behaviour of the Lempel-Ziv parsing scheme in digital search trees*, Theoret. Comput. Sci., 144 (1995), pp. 161–197.

[8] T. Kawabata, *Exact analysis of the Lempel-Ziv algorithm for I. I. D. sources*, IEEE Trans. Inform. Theory, 39 (1993), pp. 698–708.

[9] G. Louchard and W. Szpankowski, *On the average redundancy rate of the Lempel-Ziv code*, IEEE Trans. Inform. Theory, 43 (1997), pp. 2–8.

[10] H. Morita and K. Kobayashi, *On asymptotic optimality of a sliding window variation of Lempel-Ziv codes*, IEEE Trans. Inform. Theory, 39 (1993), pp. 1840–1846.

[11] E. Plotnik, M. Weinberger, and J. Ziv, *Upper bounds on the probability of sequences emitted by finite-state sources and on the redundancy of the Lempel-Ziv algorithm*, IEEE Trans. Inform. Theory, 38 (1992), pp. 66–72.

[12] M. Rodeh, V. Pratt, and S. Even, *Linear algorithm for data compression via string matching*, J. ACM, 28 (1981), pp. 16–24.

[13] D. Salomon, *Data Compression: The Complete Reference*, Springer-Verlag, New York, 1997.

[14] S. Savari, *Redundancy of the Lempel-Ziv incremental parsing rule*, IEEE Trans. Inform. Theory, 43 (1997), pp. 9–21.

[15] S. Savari, *Redundancy of the Lempel-Ziv string matching code*, IEEE Trans. Inform. Theory, 44 (1998), pp. 787–791.

[16] D. Sheinwald, *On the Ziv-Lempel proof and related topics*, Proc. IEEE, 82 (1994), pp. 866–871.

[17] W. Szpankowski, *Asymptotic properties of data compression and suffix trees*, IEEE Trans. Inform. Theory, 39 (1993), pp. 1647–1659.

[18] A. Wyner and A. Wyner, *Improved redundancy of a version of the Lempel-Ziv algorithm*, IEEE Trans. Inform. Theory, 41 (1995), pp. 723–731.

[19] A. Wyner and J. Ziv, *Fixed data base version of the Lempel-Ziv data compression algorithm*, IEEE Trans. Inform. Theory, 37 (1991), pp. 878–880.

[20] A. Wyner and J. Ziv, *The sliding-window Lempel-Ziv algorithm is asymptotically optimal*, Proc. IEEE, 82 (1994), pp. 872–877.

[21] A. J. Wyner, *The redundancy and distribution of the phrase lengths in the fixed-database Lempel-Ziv algorithm*, IEEE Trans. Inform. Theory, 43 (1997), pp. 1452–1464.

[22] E. Yang and J. Kieffer, *On the performance of data compression algorithms based on string matching*, IEEE Trans. Inform. Theory, 44 (1998), pp. 47–65.

[23] J. Ziv and A. Lempel, *A universal algorithm for sequential data compression*, IEEE Trans. Inform. Theory, 23 (1977), pp. 337–343.

[24] J. Ziv and A. Lempel, *Compression of individual sequences via variable-rate coding*, IEEE Trans. Inform. Theory, 24 (1978), pp. 530–536.