Sublinear Algorithms for Approximating String Compressibility[⋆]

Sofya Raskhodnikova[1⋆⋆], Dana Ron[2⋆⋆⋆], Ronitt Rubinfeld[3], and Adam Smith[1†]

[1] Pennsylvania State University, USA, {sofya,asmith}@cse.psu.edu
[2] Tel Aviv University, Israel, danar@eng.tau.ac.il
[3] MIT, Cambridge MA, USA, ronitt@csail.mit.edu

Abstract. We raise the question of approximating the compressibility of a string with respect to a fixed compression scheme, in sublinear time. We study this question in detail for two popular lossless compression schemes: run-length encoding (RLE) and Lempel-Ziv (LZ), and present sublinear algorithms for approximating compressibility with respect to both schemes. We also give several lower bounds that show that our algorithms for both schemes cannot be improved significantly.
Our investigation of LZ yields results whose interest goes beyond the initial questions we set out to study. In particular, we prove combinatorial structural lemmas that relate the compressibility of a string with respect to Lempel-Ziv to the number of distinct short substrings contained in it. In addition, we show that approximating the compressibility with respect to LZ is related to approximating the support size of a distribution.

1 Introduction

Given an extremely long string, it is natural to wonder how compressible it is. This fundamental question is of interest to a wide range of areas of study, including computational complexity theory, machine learning, storage systems, and communications. As massive data sets are now commonplace, the ability to estimate their compressibility with extremely efficient, even sublinear time, algorithms, is gaining in importance. The most general measure of compressibility, Kolmogorov complexity, is not computable (see [14] for a textbook treatment), nor even approximable. Even under restrictions which make it computable (such as a bound on the running time of decompression), it is probably hard to approximate in polynomial time, since an approximation would allow distinguishing random from pseudorandom strings and, hence, inverting one-way functions.

[⋆] A full version of this paper is available [17]. These results appeared previously as part of a technical report [16].

[⋆⋆] Research done while at the Hebrew University of Jerusalem, Israel, supported by the Lady Davis Fellowship, and while at the Weizmann Institute of Science, Israel.

[⋆⋆⋆] Supported by the Israel Science Foundation (grant number 89/05).

[†] Research done while at the Weizmann Institute of Science, Israel, supported by the Louis L. and Anita M. Perlman Postdoctoral Fellowship.

However, the question of how compressible a large string is with respect to a *specific compression scheme* may be tractable, depending on the particular scheme.

We raise the question of approximating the compressibility of a string with respect to a fixed compression scheme, in sublinear time, and give algorithms and nearly matching lower bounds for several versions of the problem. While this question is new, for one compression scheme, answers follow from previous work. Namely, compressibility under Huffman encoding is determined by the entropy of the symbol frequencies. Batu *et al.* [3] and Brautbar and Samorodnitsky [5] study the problem of approximating the entropy of a distribution from a small number of samples, and their results immediately imply algorithms and lower bounds for approximating compressibility under Huffman encoding.

In this work we study the compressibility approximation question in detail for two popular lossless compression schemes: run-length encoding (RLE) and Lempel-Ziv (LZ) [19]. In the RLE scheme, each run, or a sequence of consecutive occurrences of the same character, is stored as a pair: the character, and the length of the run. Run-length encoding is used to compress black and white images, faxes, and other simple graphic images, such as icons and line drawings, which usually contain many long runs. In the LZ scheme[4], a left-to-right pass of the input string is performed and at each step, the longest sequence of characters that has started in the previous portion of the string is replaced with the pointer to the previous location and the length of the sequence (for a formal definition, see Section 4). The LZ scheme and its variants have been studied extensively in machine learning and information theory, in part because they compress strings generated by an ergodic source to the shortest possible representation (given by the entropy) in the asymptotic limit (cf. [10]). Many popular archivers, such as gzip, use variations on the LZ scheme. In this work we present sublinear algorithms and corresponding lower bounds for approximating compressibility with respect to both schemes, RLE and LZ.

Motivation. Computing the compressibility of a large string with respect to specific compression schemes may be done in order to decide whether or not to compress the file, to choose which compression method is the most suitable, or check whether a small modification to the file (e.g., a rotation of an image) will make it significantly more compressible[5]. Moreover, compression schemes are used as tools for measuring properties of strings such as similarity and entropy. As such, they are applied widely in data-mining, natural language processing and genomics (see, for example, Lowenstern *et al.* [15], Kukushkina *et al.* [11], Benedetto *et al.* [4], Li *et al.* [13] and Calibrasi and Vitányi [8, 9]). In these applications, one typically needs only the *length* of the compressed version of a file, not the output itself. For example, in the clustering algorithm of [8], the distance

[4] We study the variant known as LZ77 [19], which achieves the best compressibility. There are several other variants that do not compress some inputs as well, but can be implemented more efficiently.

[5] For example, a variant of the RLE scheme, typically used to compress images, runs RLE on the concatenated rows of the image and on the concatenated columns of the image, and stores the shorter of the two compressed files.

between two objects x and y is given by a normalized version of the length of their compressed concatenation $x \| y$. The algorithm first computes all pairwise distances, and then analyzes the resulting distance matrix. This requires $\Theta(t^2)$ runs of a compression scheme, such as gzip, to cluster t objects. Even a weak approximation algorithm that can quickly rule out very incompressible strings would reduce the running time of the clustering computations dramatically.

Multiplicative and Additive Approximations. We consider three approximation notions: additive, multiplicative, and the combination of additive and multiplicative. On the input of length n, the quantities we approximate range from 1 to n. An *additive approximation* algorithm is allowed an additive error of ϵn, where $\epsilon \in (0,1)$ is a parameter. The output of a *multiplicative approximation* algorithm is within a factor $A > 1$ of the correct answer. The combined notion allows both types of error: the algorithm should output an estimate $\widehat{C}$ of the compression cost C such that $\frac{C}{A} - \epsilon n \le \widehat{C} \le A \cdot C + \epsilon n$. Our algorithms are randomized, and for all inputs the approximation guarantees hold with probability at least $\frac{2}{3}$.

We are interested in sublinear approximation algorithms, which read few positions of the input strings. For the schemes we study, purely multiplicative approximation algorithms must read almost the entire input. Nevertheless, algorithms with additive error guarantees, or a possibility of both multiplicative and additive error are often sufficient for distinguishing very compressible inputs from inputs that are not well compressible. For both the RLE and LZ schemes, we give algorithms with combined multiplicative and additive error that make few queries to the input. When it comes to additive approximations, however, the two schemes differ sharply: sublinear additive approximations are possible for the RLE compressibility, but not for LZ compressibility.

1.1 Results for Run-Length Encoding

For RLE, we present sublinear algorithms for all three approximation notions defined above, providing a trade-off between the quality of approximation and the running time. The algorithms that allow an additive approximation run in time independent of the input size. Specifically, an ϵn-additive estimate can be obtained in time[6] $\tilde{O}(1/\epsilon^3)$, and a combined estimate, with a multiplicative error of 3 and an additive error of ϵn, can be obtained in time $\tilde{O}(1/\epsilon)$. As for a strict multiplicative approximation, we give a simple 4-multiplicative approximation algorithm that runs in expected time $\tilde{O}(\frac{n}{C_{\text{rle}}(w)})$ where $C_{\text{rle}}(w)$ denotes the compression cost of the string w. For any $\gamma > 0$, the multiplicative error can be improved to $1 + \gamma$ at the cost of multiplying the running time by $\text{poly}(1/\gamma)$. Observe that the algorithm is more efficient when the string is less compressible, and less efficient when the string is more compressible. One of our lower bounds justifies such a behavior and, in particular, shows that a constant factor approximation requires linear time for strings that are very compressible. We also give a lower bound of $\Omega(1/\epsilon^2)$ for ϵn-additive approximation.

[6] The notation $\tilde{O}(g(k))$ for a function g of a parameter k means $O(g(k) \cdot \text{polylog}(g(k)))$ where $\text{polylog}(g(k)) = \log^c(g(k))$ for some constant c.

1.2 Results for Lempel-Ziv

We prove that approximating compressibility with respect to LZ is closely related to the following problem, which we call Colors: *Given access to a string τ of length n over alphabet Ψ, approximate the number of* distinct *symbols ("colors") in τ.* This is essentially equivalent to estimating the support size of a distribution [18]. Variants of this problem have been considered under various guises in the literature: in databases it is referred to as approximating distinct values (Charikar *et al.* [7]), in statistics as estimating the number of species in a population (see the over 800 references maintained by Bunge [6]), and in streaming as approximating the frequency moment F_0 (Alon *et al.* [1], Bar-Yossef *et al.* [2]). Most of these works, however, consider models different from ours. For our model, there is an A-multiplicative approximation algorithm of [7], that runs in time $O\left(\frac{n}{A^2}\right)$, matching the lower bound in [7, 2]. There is also an almost linear lower bound for approximating Colors with additive error [18].

We give a reduction from LZ compressibility to Colors and vice versa. These reductions allow us to employ the known results on Colors to give algorithms and lower bounds for this problem. Our approximation algorithm for LZ compressibility combines a multiplicative and additive error. The running time of the algorithm is $\tilde{O}\left(\frac{n}{A^3\epsilon}\right)$ where A is the multiplicative error and ϵn is the additive error. In particular, this implies that for any $\alpha > 0$, we can distinguish, in sublinear time $\tilde{O}(n^{1-\alpha})$, strings compressible to $O(n^{1-\alpha})$ symbols from strings only compressible to $\Omega(n)$ symbols.[7]

The main tool in the algorithm consists of two combinatorial structural lemmas that relate compressibility of the string to the number of distinct short substrings contained in it. Roughly, they say that a string is well compressible with respect to LZ if and only if it contains few distinct substrings of length ℓ for all small ℓ (when considering all $n - \ell + 1$ possible overlapping substrings). The simpler of the two lemmas was inspired by a structural lemma for grammars by Lehman and Shelat [12]. The combinatorial lemmas allow us to establish a reduction from LZ compressibility to Colors and employ a (simple) algorithm for approximating Colors in our algorithm for LZ.

Interestingly, we can show that there is also a reduction in the *opposite direction*: namely, approximating Colors reduces to approximating LZ compressibility. The lower bound of [18], combined with the reduction from Colors to LZ, implies that our algorithm for LZ cannot be improved significantly. In particular, our lower bound implies that for any $B = n^{o(1)}$, distinguishing strings compressible by LZ to $\tilde{O}(n/B)$ symbols from strings compressible to $\tilde{\Omega}(n)$ symbols requires $n^{1-o(1)}$ queries.

1.3 Further Research

It would be interesting to extend our results for estimating the compressibility under LZ77 to other variants of LZ, such as dictionary-based LZ78 [20]. Compressibility under LZ78 can be drastically different from compressibility under

[7] To see this, set $A = o(n^{\alpha/2})$ and $\epsilon = o(n^{-\alpha/2})$.

LZ77: e.g., for 0^n they differ roughly by a factor of $\sqrt{n}$. Another open question is approximating compressibility for schemes other than RLE and LZ. In particular, it would be interesting to design approximation algorithms for lossy compression schemes such as JPEG, MPEG and MP3. One lossy compression scheme to which our results extend directly is Lossy RLE, where some characters, e.g., the ones that represent similar colors, are treated as the same character.

1.4 Organization

We start with some definitions in Section 2. Section 3 contains our results for RLE. Section 4 deals with the LZ scheme. All missing details (descriptions of algorithms and proofs of claims) can be found in [17].

2 Preliminaries

The input to our algorithms is usually a string w of length n over a finite alphabet Σ. The quantities we approximate, such as compression cost of w under a specific algorithm, range from 1 to n. We consider estimates to these quantities that have both multiplicative and additive error. We call $\widehat{C}$ an (λ, ϵ)-*estimate* for C if $\frac{C}{\lambda} - \epsilon n \;\le\; \widehat{C} \;\le\; \lambda \cdot C + \epsilon n$, and say an algorithm (λ, ϵ)-*estimates* C (or is an (λ, ϵ)-*approximation algorithm* for C) if for each input it produces an (λ, ϵ)-estimate for C with probability at least $\frac{2}{3}$.

When the error is purely additive or multiplicative, we use the following shorthand: ϵn-*additive estimate* stands for $(1, \epsilon)$-*estimate* and λ-*multiplicative estimate*, or λ-*estimate*, stands for $(\lambda, 0)$-*estimate*. An algorithm computing an ϵn-additive estimate with probability at least $\frac{2}{3}$ is an ϵn-*additive approximation algorithm*, and if it computes an λ-multiplicative estimate then it is an λ-*multiplicative approximation algorithm*, or λ-*approximation algorithm*.

For some settings of parameters, obtaining a valid estimate is trivial. For a quantity in $[1, n]$, for example, $\frac{n}{2}$ is an $\frac{n}{2}$-additive estimate, $\sqrt{n}$ is a $\sqrt{n}$-estimate and ϵn is an (λ, ϵ)-estimate whenever $\lambda \ge \frac{1}{2\epsilon}$.

3 Run-Length Encoding

Every n-character string w over alphabet Σ can be partitioned into maximal runs of identical characters of the form σ^{ℓ}, where σ is a symbol in Σ and ℓ is the length of the run, and consecutive runs are composed of different symbols. In the *Run-Length Encoding* of w, each such run is replaced by the pair (σ, ℓ). The number of bits needed to represent such a pair is $\lceil \log(\ell + 1) \rceil + \lceil \log |\Sigma| \rceil$ plus the overhead which depends on how the separation between the characters and the lengths is implemented. One way to implement it is to use prefix-free encoding for lengths. For simplicity we ignore the overhead in the above expression, but our analysis can be adapted to any implementation choice. The *cost of the run-length encoding*, denoted by $C_{\mathrm{rle}}(w)$, is the sum over all runs of $\lceil \log(\ell + 1) \rceil + \lceil \log |\Sigma| \rceil$.

3.1 An ϵn-Additive Estimate with $\tilde{O}(1/\epsilon^3)$ Queries

Our first algorithm for approximating the cost of RLE is very simple: it samples a few positions in the input string uniformly at random and bounds the lengths of the runs to which they belong by looking at the positions to the left and to the right of each sample. If the corresponding run is short, its length is established exactly; if it is long, we argue that it does not contribute much to the encoding cost. For each index $t \in [n]$, let $\ell(t)$ be the length of the run to which w_t belongs. The cost contribution of index t is defined as

$$c(t) = \frac{\lceil \log(\ell(t)+1) \rceil + \lceil \log |\Sigma| \rceil}{\ell(t)}. \tag{1}$$

By definition, $\dfrac{C_{\mathrm{rle}}(w)}{n} = \mathop{\mathsf{E}}\limits_{t \in [n]}[c(t)]$, where $\mathsf{E}_{t \in [n]}$ denotes expectation over a uniformly random choice of t. The algorithm, presented below, estimates the encoding cost by the average of the cost contributions of the sampled short runs, multiplied by n.

Algorithm I: An ϵn-additive Approximation for $C_{\mathrm{rle}}(w)$

```
1. Select $q = \Theta\left(\frac{1}{\epsilon^2}\right)$ indices $t_1, \ldots, t_q$ uniformly and independently at random.
2. For each $i \in [q]$ :
   (a) Query $t_i$ and up to $\ell_0 = \frac{8\log(4|\Sigma|/\epsilon)}{\epsilon}$ positions in its vicinity to bound $\ell(t_i)$.
   (b) Set $\hat{c}(t_i) = c(t_i)$ if $\ell(t_i) < \ell_0$ and $\hat{c}(t_i) = 0$ otherwise.
3. Output $\widehat{C}_{\mathrm{rle}} = n \cdot \mathop{\mathsf{E}}\limits_{i \in [q]}[\hat{c}(t_i)]$.
```

Correctness. We first prove that the algorithm is an ϵn-additive approximation. The error of the algorithm comes from two sources: from ignoring the contribution of long runs and from sampling. The ignored indices t, for which $\ell(t) \geq \ell_0$, do not contribute much to the cost. Since the cost assigned to the indices monotonically decreases with the length of the run to which they belong, for each such index,

$$c(t) \leq \frac{\lceil \log(\ell_0+1) \rceil + \lceil \log |\Sigma| \rceil}{\ell_0} \leq \frac{\epsilon}{2}. \tag{2}$$

Therefore,

$$\frac{C_{\mathrm{rle}}(w)}{n} - \frac{\epsilon}{2} \quad \leq \quad \frac{1}{n} \cdot \sum_{t:\, \ell(t) < \ell_0} c(t) \quad \leq \quad \frac{C_{\mathrm{rle}}(w)}{n}. \tag{3}$$

Equivalently, $\frac{C_{\mathrm{rle}}(w)}{n} - \frac{\epsilon}{2} \leq \mathsf{E}_{i \in [n]}[\hat{c}(t_i)] \leq \frac{C_{\mathrm{rle}}(w)}{n}$.

By an additive Chernoff bound, with high constant probability, the sampling error in estimating $\mathsf{E}[\hat{c}(t_i)]$ is at most $\epsilon/2$. Therefore, $\widehat{C}_{\mathrm{rle}}$ is an ϵn-additive estimate of $C_{\mathrm{rle}}(w)$, as desired.

Query and time complexity. (Assuming $|\Sigma|$ is constant.) Since the number of queries performed for each selected t_i is $O(\ell_0) = O(\log(1/\epsilon)/\epsilon)$, the total number of queries, as well as the running time, is $O(\log(1/\epsilon)/\epsilon^3)$.

3.2 Summary of Positive Results on RLE

After stating Theorem 1 that summarizes our positive results, we briefly discuss some of the ideas used in the algorithms omitted from this version of the paper.

Theorem 1 *Let $w \in \Sigma^n$ be a string to which we are given query access.*

1. *Algorithm I gives ϵn-additive approximation to $C_{\mathrm{rle}}(w)$ in time $\tilde{O}(1/\epsilon^3)$.*
2. *$C_{\mathrm{rle}}(w)$ can be $(3, \epsilon)$-estimated in time $\tilde{O}(1/\epsilon)$.*
3. *$C_{\mathrm{rle}}(w)$ can be 4-estimated in expected time $\tilde{O}\left(\frac{n}{C_{\mathrm{rle}}(w)}\right)$. A $(1+\gamma)$-estimate of $C_{\mathrm{rle}}(w)$ can be obtained in expected time $\tilde{O}\left(\frac{n}{C_{\mathrm{rle}}(w)} \cdot \mathrm{poly}(1/\gamma)\right)$. The algorithm needs no prior knowledge of $C_{\mathrm{rle}}(w)$.*

Section 3.1 gives a complete proof of Item 1. The algorithm in Item 2 partitions the positions in the string into *buckets* according to the length of the runs they belong to. It estimates the sizes of different buckets with different precision, depending on the size of the bucket and the length of the runs it contains. The main idea in Item 3 is to search for $C_{\mathrm{rle}}(w)$, using the algorithm from Item 2 repeatedly (with different parameters) to establish successively better estimates.

3.3 Lower Bounds for RLE

We give two lower bounds, for multiplicative and additive approximation, respectively, which establish that the running times in Items 1 and 3 of Theorem 1 are essentially tight.

Theorem 2 *1. For all $A > 1$, any A-approximation algorithm for C_{rle} requires $\Omega\left(\frac{n}{A^2 \log n}\right)$ queries. Furthermore, if the input is restricted to strings with compression cost $C_{\mathrm{rle}}(w) \geq C$, then $\Omega\left(\frac{n}{CA^2 \log(n)}\right)$ queries are necessary.*

2. For all $\epsilon \in \left(0, \frac{1}{2}\right)$, any ϵn-additive approximation algorithm for C_{rle} requires $\Omega(1/\epsilon^2)$ queries.

A Multiplicative Lower Bound (Proof of Theorem 2, Item 1): The claim follows from the next lemma:

Lemma 3 *For every $n \geq 2$ and every integer $1 \leq k \leq n/2$, there exists a family of strings, denoted W_k, for which the following holds: (1) $C_{\mathrm{rle}}(w) = \Theta\left(k \log(\frac{n}{k})\right)$ for every $w \in W_k$; (2) Distinguishing a uniformly random string in W_k from one in $W_{k'}$, where $k' > k$, requires $\Omega\left(\frac{n}{k'}\right)$ queries.*

Proof: Let $\Sigma = \{0, 1\}$ and assume for simplicity that n is divisible by k. Every string in W_k consists of k blocks, each of length $\frac{n}{k}$. Every odd block contains only 1s and every even block contains a single 0. The strings in W_k differ in the locations of the 0s within the even blocks. Every $w \in W_k$ contains $k/2$ isolated 0s and $k/2$ runs of 1s, each of length $\Theta(\frac{n}{k})$. Therefore, $C_{\mathrm{rle}}(w) = \Theta\left(k \log(\frac{n}{k})\right)$. To distinguish a random string in W_k from one in $W_{k'}$ with probability 2/3, one must make $\Omega(\frac{n}{\max(k,k')})$ queries since, in both cases, with asymptotically fewer queries the algorithm sees only 1's with high probability. ■

Additive Lower Bound (Proof Theorem 2, Item 1): For any $p \in [0,1]$ and sufficiently large n, let $\mathcal{D}_{n,p}$ be the following distribution over n-bit strings. For simplicity, consider n divisible by 3. The string is determined by $\frac{n}{3}$ independent coin flips, each with bias p. Each "heads" extends the string by three runs of length 1, and each "tails", by a run of length 3. Given the sequence of run lengths, dictated by the coin flips, output the unique binary string that starts with 0 and has this sequence of run lengths.[8]

Let W be a random variable drawn according to $\mathcal{D}_{n,1/2}$ and W', according to $\mathcal{D}_{n,1/2+\epsilon}$. The following facts are established in the full version [17]: (a) $\Omega(1/\epsilon^2)$ queries are necessary to reliably distinguish W from W', and (b) With high probability, the encoding costs of W and W' differ by $\Omega(\epsilon n)$. Together these facts imply the lower bound. $\blacksquare$

4 Lempel Ziv Compression

In this section we consider a variant of Lempel and Ziv's compression algorithm [19], which we refer to as LZ77. In all that follows we use the shorthand $[n]$ for $\{1,\ldots,n\}$. Let $w \in \Sigma^n$ be a string over an alphabet Σ. Each symbol of the compressed representation of w, denoted $LZ(w)$, is either a character $\sigma \in \Sigma$ or a pair (p,ℓ) where $p \in [n]$ is a pointer (index) to a location in the string w and ℓ is the length of the substring of w that this symbol represents. To compress w, the algorithm works as follows. Starting from $t=1$, at each step the algorithm finds the longest substring $w_t \ldots w_{t+\ell-1}$ for which there exists an index $p < t$, such that $w_p \ldots w_{p+\ell-1} = w_t \ldots w_{t+\ell-1}$. (The substrings $w_p \ldots w_{p+\ell-1}$ and $w_t \ldots w_{t+\ell-1}$ may overlap.) If there is no such substring (that is, the character w_t has not appeared before) then the next symbol in $LZ(w)$ is w_t, and $t = t+1$. Otherwise, the next symbol is (p,ℓ) and $t = t+\ell$. We refer to the substring $w_t \ldots w_{t+\ell-1}$ (or w_t when w_t is a new character) as a *compressed segment*.

Let $C_{\mathrm{LZ}}(w)$ denote the number of symbols in the compressed string $LZ(w)$. (We do not distinguish between symbols that are characters in Σ, and symbols that are pairs (p,ℓ).) Given query access to a string $w \in \Sigma^n$, we are interested in computing an estimate $\widehat{C}_{\mathrm{LZ}}$ of $C_{\mathrm{LZ}}(w)$. As we shall see, this task reduces to estimating the number of distinct substrings in w of different lengths, which in turn reduces to estimating the number of distinct characters ("colors") in a string. The actual length of the binary representation of the compressed substring is at most a factor of $2\log n$ larger than $C_{\mathrm{LZ}}(w)$. This is relatively negligible given the quality of the estimates that we can achieve in sublinear time.

We begin by relating LZ compressibility to Colors (§4.1), then use this relation to discuss algorithms (§4.2) and lower bounds (§4.3) for compressiblity.

[8] Let b_i be a boolean variable representing the outcome of the ith coin. Then the output is $0b_1 01\overline{b_2}10b_3 01\overline{b_4}1\ldots$

4.1 Structural Lemmas

Our algorithm for approximating the compressibility of an input string with respect to LZ77 uses an approximation algorithm for Colors (defined in the introduction) as a subroutine. The main tool in the reduction from LZ77 to Colors is the relation between $C_{\mathrm{LZ}}(w)$ and the number of distinct substrings in w, formalized in the two structural lemmas. In what follows, $d_\ell(w)$ denotes the *number of distinct substrings* of length ℓ in w. Unlike compressed segments in w, which are disjoint, these substrings may overlap.

Lemma 4 (Structural Lemma 1) *For every $\ell \in [n]$, $C_{LZ}(w) \geq \frac{d_\ell(w)}{\ell}$.*

Lemma 5 (Structural lemma 2) *Let $\ell_0 \in [n]$. Suppose that for some integer m and for every $\ell \in [\ell_0]$, $d_\ell(w) \leq m \cdot \ell$. Then $C_{LZ}(w) \leq 4(m \log \ell_0 + n/\ell_0)$.*

Proof of Lemma 4. This proof is similar to the proof of a related lemma concerning grammars from [12]. First note that the lemma holds for $\ell = 1$, since each character w_t in w that has not appeared previously (that is, $w_{t'} \neq w_t$ for every $t' < t$) is copied by the compression algorithm to $LZ(w)$.

For the general case, fix $\ell > 1$. Recall that $w_t \dots w_{t+k-1}$ of w is a *compressed segment* if it is represented by one symbol (p, k) in $LZ(w)$. Any substring of lenth ℓ that occurs *within* a compressed segment must have occurred previously in the string. Such substrings can be ignored for our purposes: the number of *distinct* length-ℓ substrings is bounded above by the number of length-ℓ substrings that start inside one compressed segment and end in another. Each segment (except the last) contributes $(\ell - 1)$ such substrings. Therefore, $d_\ell(w) \leq (C_{\mathrm{LZ}}(w) - 1)(\ell - 1) < C_{\mathrm{LZ}}(w) \cdot \ell$ for every $\ell > 1$. ■

Proof of Lemma 5. Let $n_\ell(w)$ denote the number of compressed segments of length ℓ in w, not including the last compressed segment. We use the shorthand n_ℓ for $n_\ell(w)$ and d_ℓ for $d_\ell(w)$. In order to prove the lemma we shall show that for every $1 \leq \ell \leq \lfloor \ell_0/2 \rfloor$,

$$\sum_{k=1}^{\ell} n_k \leq 2(m+1) \cdot \sum_{k=1}^{\ell} \frac{1}{k} \,. \tag{4}$$

For all $\ell \geq 1$, since the compressed segments in w are disjoint, $\sum_{k=\ell+1}^{n} n_k \leq \frac{n}{\ell+1}$. If we substitute $\ell = \lfloor \ell_0/2 \rfloor$ in the last two equations and sum them up, we get:

$$\sum_{k=1}^{n} n_k \leq 2(m+1) \cdot \sum_{k=1}^{\lfloor \ell_0/2 \rfloor} \frac{1}{k} + \frac{2n}{\ell_0} \leq 2(m+1)(\ln \ell_0 + 1) + \frac{2n}{\ell_0}. \tag{5}$$

Since $C_{\mathrm{LZ}}(w) = \sum_{k=1}^{n} n_k + 1$, the lemma follows.

It remains to prove Equation (4). We do so below by induction on ℓ, using the following claim.

Claim 6 *For every* $1 \leq \ell \leq \lfloor \ell_0/2 \rfloor$, $\displaystyle\sum_{k=1}^{\ell} k \cdot n_k \leq 2\ell(m+1)$.

Proof: We show that each position $j \in \{\ell, \ldots, n-\ell\}$ that participates in a compressed substring of length at most ℓ in w can be mapped to a distinct length-2ℓ substring of w. Since $\ell \leq \ell_0/2$, by the premise of the lemma, there are at most $2\ell \cdot m$ distinct length-2ℓ substrings. In addition, the first $\ell - 1$ and the last ℓ positions contribute less than 2ℓ symbols. The claim follows.

We call a substring *new* if no instance of it started in the previous portion of w. Namely, $w_t \ldots w_{t+\ell-1}$ is *new* if there is no $p < t$ such that $w_t \ldots w_{t+\ell-1} = w_p \ldots w_{p+\ell-1}$. Consider a compressed substring $w_t \ldots w_{t+k-1}$ of length $k \leq \ell$. The substrings of length greater than k that start at w_t must be *new*, since LZ77 finds the longest substring that appeared before. Furthermore, every substring that contains such a *new* substring is also *new*. That is, every substring $w_{t'} \ldots w_{t+k'}$ where $t' \leq t$ and $k' \geq k + (t' - t)$, is *new*.

Map each position $j \in \{\ell, \ldots, n-\ell\}$ in the compressed substring $w_t \ldots w_{t+k-1}$ to the length-2ℓ substring that ends at $w_{j+\ell}$. Then each position in $\{\ell, \ldots, n-\ell\}$ that appears in a compressed substring of length at most ℓ is mapped to a distinct length-2ℓ substring, as desired. ■ (Claim 6)

Establishing Equation (4). We prove Equation (4) by induction on ℓ. Claim 6 with ℓ set to 1 gives the base case, i.e., $n_1 \leq 2(m+1)$. For the induction step, assume the induction hypothesis for every $j \in [\ell - 1]$. To prove it for ℓ, add the equation in Claim 6 to the sum of the induction hypothesis inequalities (Equation (4)) for every $j \in [\ell - 1]$. The left hand side of the resulting inequality is

$$\begin{aligned}\sum_{k=1}^{\ell} k \cdot n_k + \sum_{j=1}^{\ell-1} \sum_{k=1}^{j} n_k &= \sum_{k=1}^{\ell} k \cdot n_k + \sum_{k=1}^{\ell-1} \sum_{j=1}^{\ell-k} n_k \\ &= \sum_{k=1}^{\ell} k \cdot n_k + \sum_{k=1}^{\ell-1} (\ell - k) \cdot n_k = \ell \cdot \sum_{k=1}^{\ell} n_k \,.\end{aligned}$$

The right hand side, divided by the factor $2(m+1)$, which is common to all inequalities, is

$$\ell + \sum_{j=1}^{\ell-1} \sum_{k=1}^{j} \frac{1}{k} = \ell + \sum_{k=1}^{\ell-1} \sum_{j=1}^{\ell-k} \frac{1}{k} = \ell + \sum_{k=1}^{\ell-1} \frac{\ell - k}{k} = \ell + \ell \cdot \sum_{k=1}^{\ell-1} \frac{1}{k} - (\ell - 1) = \ell \cdot \sum_{k=1}^{\ell} \frac{1}{k} \,.$$

Dividing both sides by ℓ gives the inequality in Equation (4). ■ (Lemma 5)

4.2 An Algorithm for LZ77

This subsection describes an algorithm for approximating the compressibility of an input string with respect to LZ77, which uses an approximation algorithm for

Colors as a subroutine. The main tool in the reduction from LZ77 to Colors consists of structural lemmas 4 and 5, summarized in the following corollary.

Corollary 7 *For any $\ell_0 \geq 1$, let $m = m(\ell_0) = \max_{\ell=1}^{\ell_0} \frac{d_\ell(w)}{\ell}$. Then*

$$m \;\; \leq \;\; C_{LZ}(w) \;\; \leq \;\; 4 \cdot \left(m \log \ell_0 + \frac{n}{\ell_0} \right) \; .$$

The corollary allows us to approximate C_{LZ} from estimates for d_ℓ for all $\ell \in [\ell_0]$. To obtain these estimates, we use the algorithm of [7] for Colors as a subroutine (in the full version [17] we also describe a simpler Colors algorithm with the same provable guarantees). Recall that an algorithm for Colors approximates the number of distinct colors in an input string, where the ith character represents the ith color. We denote the number of colors in an input string τ by $C_{\mathrm{COL}}(\tau)$. To approximate d_ℓ, the number of distinct length-ℓ substrings in w, using an algorithm for Colors, view each length-ℓ substring as a separate color. Each query of the algorithm for Colors can be implemented by ℓ queries to w.

Let Estimate(ℓ, B, δ) be a procedure that, given access to w, an index $\ell \in [n]$, an approximation parameter $B = B(n, \ell) > 1$ and a confidence parameter $\delta \in [0, 1]$, computes a B-estimate for d_ℓ with probability at least $1 - \delta$. It can be implemented using an algorithm for Colors, as described above, and employing standard amplification techniques to boost success probability from $\frac{2}{3}$ to $1 - \delta$: running the basic algorithm $\Theta(\log \delta^{-1})$ times and outputting the median. Since the algorithm of [7] requires $O(n/B^2)$ queries, the query complexity of Estimate(ℓ, B, δ) is $O\left(\frac{n}{B^2}\, \ell \log \delta^{-1}\right)$. Using Estimate$(\ell, B, \delta)$ as a subroutine, we get the following approximation algorithm for the cost of LZ77.

Algorithm II: An (A, ϵ)-approximation for $C_{\mathrm{LZ}}(w)$

```
1. Set $\ell_0 = \lceil \frac{2}{A\epsilon} \rceil$ and $B = \frac{A}{2\sqrt{\log(2/(A\epsilon))}}$.
2. For all $\ell$ in $[\ell_0]$, let $\hat{d}_\ell = \text{Estimate}(\ell, B, \frac{1}{3\ell_0})$.
3. Combine the estimates to get an approximation of $m$ from Corollary 7: set $\hat{m} = \max_{\ell} \frac{\hat{d}_\ell}{\ell}$.
4. Output $\widehat{C}_{\mathrm{LZ}} = \hat{m} \cdot \frac{A}{B} + \epsilon n$.
```

Theorem 8 *Algorithm II (A, ϵ)-estimates $C_{LZ}(w)$. With a proper implementation that reuses queries and an appropriate data structure, its query and time complexity are $\tilde{O}\left(\frac{n}{A^3 \epsilon}\right)$.*

Proof: By the Union Bound, with probability $\geq \frac{2}{3}$, all values $\hat{d}_\ell$ computed by the algorithm are B-estimates for the corresponding d_ℓ. When this holds, $\hat{m}$ is a B-estimate for m from Corollary 7, which implies that

$$\frac{\hat{m}}{B} \;\; \leq \;\; C_{\mathrm{LZ}}(w) \;\; \leq \;\; 4 \cdot \left(\hat{m} B \log \ell_0 + \frac{n}{\ell_0} \right) \; .$$

Equivalently, $\dfrac{C_{\mathrm{LZ}} - 4(n/\ell_0)}{4B\log \ell_0} \le \hat{m} \le B \cdot C_{\mathrm{LZ}}$. Multiplying all three terms by $\frac{A}{B}$ and adding ϵn to them, and then substituting parameter settings for ℓ_0 and B, specified in the algorithm, shows that $\widehat{C}_{\mathrm{LZ}}$ is indeed an (A, ϵ)-estimate for C_{LZ}.

As explained before the algorithm statement, each call to Estimate$(\ell, B, \frac{1}{3\ell_0})$ costs $O\left(\frac{n}{B^2}\,\ell \log \ell_0\right)$ queries. Since the subroutine is called for all $\ell \in [\ell_0]$, the straightforward implementation of the algorithm would result in $O\left(\frac{n}{B^2}\ell_0^2 \log \ell_0\right)$ queries. Our analysis of the algorithm, however, does not rely on independence of queries used in different calls to the subroutine, since we employ the Union Bound to calculate the error probability. It will still apply if we first run Estimate to approximate d_{ℓ_0} and then reuse its queries for the remaining calls to the subroutine, as though it requested to query only the length-ℓ prefixes of the length-ℓ_0 substrings queried in the first call. With this implementation, the query complexity is $O\left(\frac{n}{B^2}\ell_0 \log \ell_0\right) = O\left(\frac{n}{A^3\epsilon} \log^2 \frac{1}{A\epsilon}\right)$. To get the same running time, one can maintain counters for all $\ell \in [\ell_0]$ for the number of distinct length-ℓ substrings seen so far and use a trie to keep the information about the queried substrings. Every time a new node at some depth ℓ is added to the trie, the ℓth counter is incremented. ■

4.3 Lower Bounds: Reducing Colors to LZ77

We have demonstrated that estimating the LZ77 compressibility of a string reduces to Colors. As shown in [18], Colors is quite hard, and it is not possible to improve much on the simple approximation algorithm in [7] , on which we base the LZ77 approximation algorithm in the previous subsection. A natural question is whether there is a better algorithm for the LZ77 estimation problem. That is, is the LZ77 estimation strictly easier than Colors? As we shall see, it is not much easier in general.

Lemma 9 (Reduction from Colors to LZ77) *Suppose there exists an algorithm $\mathcal{A}_{LZ}$ that, given access to a string w of length n over an alphabet Σ, performs $q = q(n, |\Sigma|, \alpha, \beta)$ queries and with probability at least $5/6$ distinguishes between the case that $C_{LZ}(w) \le \alpha n$ and the case that $C_{LZ}(w) > \beta n$, for some $\alpha < \beta$.*

Then there is an algorithm for Colors taking inputs of length $n' = \Theta(\alpha n)$ that performs q queries and, with probability at least $2/3$, distinguishes inputs with at most $\alpha' n'$ colors from those with at least $\beta' n'$ colors, $\alpha' = \alpha/2$ and $\beta' = \beta \cdot 2 \cdot \max\left\{1, \frac{4\log n'}{\log|\Sigma|}\right\}$.

Two notes are in place regarding the reduction. The first is that the gap between the parameters α' and β' that is required by the Colors algorithm obtained in Lemma 9, is larger than the gap between the parameters α and β for which the LZ-compressibility algorithm works, by a factor of $4 \cdot \max\left\{1, \frac{4\log n'}{\log|\Sigma|}\right\}$. In particular, for binary strings $\frac{\beta'}{\alpha'} = O\left(\log n' \cdot \frac{\beta}{\alpha}\right)$, while if the alphabet is large, say, of size at least n', then $\frac{\beta'}{\alpha'} = O\left(\frac{\beta}{\alpha}\right)$. In general, the gap increases by at most $O(\log n')$. The second note is that the number of queries, q, is a function of the parameters of the LZ-compressibility problem and, in particular, of the length of the input strings, n. Hence, when writing q as a function of the parameters of Colors and, in particular, as a function of $n' = \Theta(\alpha n)$, the complexity may be somewhat larger. It is an open question whether a reduction without such increase is possible.

Prior to proving the lemma , we discuss its implications. [18] give a strong lower bound on the sample complexity of approximation algorithms for Colors. An interesting special case is that a subpolynomial-factor approximation for Colors requires many queries even with a promise that the strings are only slightly compressible: for any $B = n^{o(1)}$, distinguishing inputs with $n/11$ colors from those with n/B colors requires $n^{1-o(1)}$ queries. Lemma 9 extends that bound to estimating LZ compressibility: *For any $B = n^{o(1)}$, and any alphabet Σ, distinguishing strings with LZ compression cost $\tilde{\Omega}(n)$ from strings with cost $\tilde{O}(n/B)$ requires $n^{1-o(1)}$ queries.*

The lower bound for Colors in [18] applies to a broad range of parameters, and yields the following general statement when combined with Lemma 9:

Corollary 10 (LZ is Hard to Approximate with Few Samples) *For sufficiently large n, all alphabets Σ and all $B \le n^{1/4}/(4 \log n^{3/2})$, there exist $\alpha, \beta \in (0,1)$ where $\beta = \Omega\left(\min\left\{1, \frac{\log|\Sigma|}{4 \log n}\right\}\right)$ and $\alpha = O\left(\frac{\beta}{B}\right)$, such that every algorithm that distinguishes between the case that $C_{LZ}(w) \le \alpha n$ and the case that $C_{LZ}(w) > \beta n$ for $w \in \Sigma^n$, must perform $\Omega\left(\left(\frac{n}{B'}\right)^{1-\frac{2}{k}}\right)$ queries for $B' = \Theta\left(B \cdot \max\left\{1, \frac{4 \log n}{\log|\Sigma|}\right\}\right)$ and $k = \Theta\left(\sqrt{\frac{\log n}{\log B' + \frac{1}{2}\log\log n}}\right)$.*

Proof of Lemma 9. Suppose we have an algorithm $\mathcal{A}_{\text{LZ}}$ for LZ-compressibility as specified in the premise of Lemma 9. Here we show how to transform a Colors instance τ into an input for $\mathcal{A}_{\text{LZ}}$, and use the output of $\mathcal{A}_{\text{LZ}}$ to distinguish τ with at most $\alpha' n'$ colors from τ with at least $\beta' n'$ colors, where α' and β' are as specified in the lemma. We shall assume that $\beta' n'$ is bounded below by some sufficiently large constant. Recall that in the reduction from LZ77 to Colors, we transformed substrings into colors. Here we perform the reverse operation.

Given a Colors instance τ of length n', we transform it into a string of length $n = n' \cdot k$ over Σ, where $k = \lceil \frac{1}{\alpha} \rceil$. We then run $\mathcal{A}_{\text{LZ}}$ on w to obtain information about τ. We begin by replacing each color in τ with a uniformly selected substring in Σ^k. The string w is the concatenation of the corresponding substrings (which we call *blocks*). We show that:

1. If τ has at most $\alpha' n'$ colors, then $C_{\text{LZ}}(w) \le 2\alpha' n$;
2. If τ has at least $\beta' n'$ colors, then $\Pr_w[C_{\text{LZ}}(w) \ge \frac{1}{2} \cdot \min\left\{1, \frac{\log|\Sigma|}{4 \log n'}\right\} \cdot \beta' n] \ge \frac{7}{8}$.

That is, in the first case we get an input w for Colors such that $C_{\text{LZ}}(w) \le \alpha n$ for $\alpha = 2\alpha'$, and in the second case, with probability at least 7/8, $C_{\text{LZ}}(w) \ge \beta n$ for

$\beta = \frac{1}{2} \cdot \min\left\{1, \frac{\log|\Sigma|}{4\log n'}\right\} \cdot \beta'$. Recall that the gap between α' and β' is assumed to be sufficiently large so that $\alpha < \beta$. To distinguish the case that $C_{\mathrm{COL}}(\tau) \le \alpha' n'$ from the case that $C_{\mathrm{COL}}(\tau) > \beta' n'$, we can run $\mathcal{A}_{\mathrm{LZ}}$ on w and output its answer. Taking into account the failure probability of $\mathcal{A}_{\mathrm{LZ}}$ and the failure probability in Item 2 above, the Lemma follows.

We prove these two claims momentarily, but first observe that in order to run the algorithm $\mathcal{A}_{\mathrm{LZ}}$, there is no need to generate the whole string w. Rather, upon each query of $\mathcal{A}_{\mathrm{LZ}}$ to w, if the index of the query belongs to a block that has already been generated, the answer to $\mathcal{A}_{\mathrm{LZ}}$ is determined. Otherwise, we query the element (color) in τ that corresponds to the block. If this color was not yet observed, then we set the block to a uniformly selected substring in Σ^k. If this color was already observed in τ, then we set the block according to the substring that was already selected for the color. In either case, the query to w can now be answered. Thus, each query to w is answered by performing at most one query to τ.

It remains to prove the two items concerning the relation between the number of colors in τ and $C_{\mathrm{LZ}}(w)$. If τ has at most $\alpha' n'$ colors then w contains at most $\alpha' n'$ distinct blocks. Since each block is of length k, at most k compressed segments start in each new block. By definition of LZ77, at most one compressed segment starts in each repeated block. Hence,

$$C_{\mathrm{LZ}}(w) \le \alpha' n' \cdot k + (1-\alpha')n' \le \alpha' n + n' \le 2\alpha' n.$$

If τ contains $\beta' n'$ or more colors, w is generated using at least $\beta' n' \cdot \log(|\Sigma|^k) = \beta' n \log|\Sigma|$ random bits. Hence, with high probability (e.g., at least 7/8) over the choice of these random bits, any lossless compression algorithm (and in particular LZ77) must use at least $\beta' n \log|\Sigma| - 3$ bits to compress w. Each symbol of the compressed version of w can be represented by $\max\{\lceil\log|\Sigma|\rceil, 2\lceil\log n\rceil\} + 1$ bits, since it is either an alphabet symbol or a pointer-length pair. Since $n = n'\lceil 1/\alpha' \rceil$, and $\alpha' > 1/n'$, each symbol takes at most $\max\{4\log n', \log|\Sigma|\} + 2$ bits to represent. This means the number of symbols in the compressed version of w is

$$C_{\mathrm{LZ}}(w) \ge \frac{\beta' n \log|\Sigma| - 3}{\max\{4\log n', \log|\Sigma|\}) + 2} \ge \frac{1}{2} \cdot \beta' n \cdot \min\left\{1, \frac{\log|\Sigma|}{4\log n'}\right\}$$

where we have used the fact that $\beta' n'$, and hence $\beta' n$, is at least some sufficiently large constant. ■

Acknowledgements. We would like to thank Amir Shpilka, who was involved in a related paper on distribution support testing [18] and whose comments greatly improved drafts of this article. We would also like to thank Eric Lehman for discussing his thesis material with us and Oded Goldreich and Omer Reingold for helpful comments.

References

1. Noga Alon, Yossi Matias, and Mario Szegedy. The space complexity of approximating the frequency moments. *J. Comput. Syst. Sci.*, 58(1):137–147, 1999.

2. Ziv Bar-Yossef, Ravi Kumar, and D. Sivakumar. Sampling algorithms: lower bounds and applications. In *Proceedings of the thirty-third annual ACM symposium on Theory of computing*, pages 266–275, New York, NY, USA, 2001. ACM Press.
3. Tugkan Batu, Sanjoy Dasgupta, Ravi Kumar, and Ronitt Rubinfeld. The complexity of approximating the entropy. *SIAM Journal on Computing*, 35(1):132–150, 2005.
4. Dario Benedetto, Emanuele Caglioti, and Vittorio Loreto. Language trees and zipping. *Phys. Rev. Lett.*, 88(4), 2002. See comment by Khmelev DV, Teahan WJ, *Phys Rev Lett.* 90(8):089803, 2003 and the reply *Phys Rev Lett.* 90(8):089804, 2003.
5. Mickey Brautbar and Alex Samorodnitsky. Approximating the entropy of large alphabets. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, 2007.
6. John Bunge. Bibligraphy on estimating the number of classes in a population. `www.stat.cornell.edu/∼bunge/bibliography.htm`.
7. Moses Charikar, Surajit Chaudhuri, Rajeev Motwani, and Vivek R. Narasayya. Towards estimation error guarantees for distinct values. In *PODS*, pages 268–279. ACM, 2000.
8. Rudi Cilibrasi and Paul M. B. Vitányi. Clustering by compression. *IEEE Transactions on Information Theory*, 51(4):1523–1545, 2005.
9. Rudi Cilibrasi and Paul M. B. Vitányi. Similarity of objects and the meaning of words. In Jin-Yi Cai, S. Barry Cooper, and Angsheng Li, editors, *TAMC*, volume 3959 of *Lecture Notes in Computer Science*, pages 21–45. Springer, 2006.
10. T. Cover and J. Thomas. *Elements of Information Theory.* Wiley & Sons, 1991.
11. O. V. Kukushkina, A. A. Polikarpov, and D. V. Khmelev. Using literal and grammatical statistics for authorship attribution. *Prob. Peredachi Inf.*, 37(2):96–98, 2000. [Probl. Inf. Transm. (Engl. Transl.) 37, 172–184 (2001)].
12. Eric Lehman and Abhi Shelat. Approximation algorithms for grammer-based compression. In *Proc. 18th Annual Symp. on Discrete Algorithms*, pages 205–212, 2002.
13. Ming Li, Xin Chen, Xin Li, Bin Ma, and Paul M. B. Vitányi. The similarity metric. *IEEE Transactions on Information Theory*, 50(12):3250–3264, 2004. Prelim. version in *SODA 2003.*
14. Ming Li and Paul Vitányi. *An Introduction to Kolmogorov Complexity and Its Applications.* Springer, 1997.
15. David Loewenstern, Haym Hirsh, Michiel Noordewier, and Peter Yianilos. DNA sequence classification using compression-based induction. Technical Report 95-04, Rutgers University, DIMACS, 1995.
16. Sofya Raskhodnikova, Dana Ron, Ronitt Rubinfeld, Amir Shpilka, and Adam Smith. Sublinear algorithms for approximating string compressibility and the distribution support size. *Electronic Colloquium on Computational Complexity*, TR05-125, 2005.
17. Sofya Raskhodnikova, Dana Ron, Ronitt Rubinfeld, and Adam Smith. Sublinear algorithms for approximating string compressibility. Full version of this paper, in preparation, Arxiv Report 0706.1084 [cs.DS], June 2007.
18. Sofya Raskhodnikova, Dana Ron, Amir Shpilka, and Adam Smith. On the difficulty of approximating the support size of a distribution. Manuscript, 2007.
19. Jacob Ziv and Abraham Lempel. A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory*, 23:337–343, 1977.
20. Jacob Ziv and Abraham Lempel. Compression of individual sequences via variable-rate coding. *IEEE Transactions on Information Theory*, 24:530–536, 1978.