Sublinear Algorithms for Approximating String Compressibility

Sofya Raskhodnikova, Dana Ron, Ronitt Rubinfeld and Adam Smith

Abstract

This work raises the question of approximating the compressibility of a string with respect to a fixed compression scheme, in sublinear time. This question is studied in detail for two popular lossless compression schemes: run-length encoding (RLE) and Lempel-Ziv (LZ). Sublinear algorithms are presented for approximating compressibility with respect to both schemes. Several lower bounds are also proven that show that our algorithms for both schemes cannot be improved significantly.

This investigation of LZ yields results whose interest goes beyond the initial questions it set out to study. In particular, it leads to 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, it is shown that approximating the compressibility with respect to LZ is related to approximating the support size of a distribution.

I. Introduction

Given an extremely long string, it is natural to wonder how compressible it is. This question is fundamental to several disciplines, including computational complexity theory, machine learning, storage systems, and communications. As massive data sets become commonplace, the ability to estimate compressibility with extremely efficient, even sublinear time, algorithms, is gaining importance. The most general measure of compressibility, Kolmogorov complexity, is not computable (see [1] 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 algorithm with non-trivial approximation guarantees would allow one to distinguish random from pseudorandom strings and, hence, invert one-way functions. Nevertheless, 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. Although this question is new, for one compression scheme, namely Huffman coding, answers follow from previous work. Compressibility under Huffman encoding is determined by the entropy of the symbol frequencies. Batu *et al.* [2] and Brautbar and Samorodnitsky [3] 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.

S. R. and A.S. are with the Department of Computer Science and Engineering, Pennsylvania State University, USA. {sofya,asmith}@cse.psu.edu. This research was done while S.R. was at the Hebrew University of Jerusalem, Israel, supported by the Lady Davis Fellowship, and while both S.R. and A.S. were at the Weizmann Institute of Science, Israel. A.S. was supported at Weizmann by the Louis L. and Anita M. Perlman Postdoctoral Fellowship.

D.R. is with Tel Aviv University, Israel. danar@eng.tau.ac.il. Supported by the Israel Science Foundation (grant number 89/05).

R.R. is with the Computer Science and AI Lab, M.I.T., Cambridge MA, USA. ronitt@csail.mit.edu.

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) [4]. 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[1], 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 IV). 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. [5]). 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.

a) 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[2]. 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.* [6], Kukushkina *et al.* [7], Benedetto *et al.* [8], Li *et al.* [9] and Calibrasi and Vitányi [10], [11]). 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 [10], the distance 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.

b) 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 $\frac{C}{A} - \epsilon n \leq \widehat{C} \leq A \cdot C + \epsilon n$. 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

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

[2]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.

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. We summarize our results in the next two sections.

A. 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[3] $\tilde{O}(1/\epsilon^3)$, 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 algorithmthat 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.

B. Results for Lempel-Ziv

We prove that approximating compressibility with respect to LZ is closely related to the following problem, which we call DISTINCT ELEMENTS (DE):

Definition 1 (DE Problem)**.** *Given access to a string τ over alphabet Ψ, approximate the number of* distinct *elements (that is, symbols) in τ.*

This is essentially equivalent to estimating the support size of a distribution [12]. 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.* [13]), in statistics as estimating the number of species in a population (see the over 800 references maintained by Bunge [14]), and in streaming as approximating the frequency moment F_0 (Alon *et al.* [15], Bar-Yossef *et al.* [16]).Most of these works, however, consider models different from ours. For our model, there is an A-multiplicative approximation algorithm of [13], that runs in time $O\left(\frac{n}{A^2}\right)$, matching the lower bound in [13], [16]. There is also an almost linear lower bound for approximating DE with additive error [12].

We give a reduction from LZ compressibility to DE and vice versa. These reductions allow us to employ the known results on DE 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

[3]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.

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.[4]

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 [17]. The combinatorial lemmas allow us to establish a reduction from LZ compressibility to DE and employ a (simple) algorithm for approximating DE in our algorithm for LZ.

Interestingly, we can show that there is also a reduction in the *opposite direction*: namely, approximating DE reduces to approximating LZ compressibility. The lower bound of [12], combined with the reduction from DE 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.

C. 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 [18]. Compressibility under LZ78 can be drastically different from compressibility under 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 scheme to which our results extend directly is a commonly used variant of RLE, where some distinct symbols, e.g., pixels of similar color, are treated as the same character.

D. Organization

We start with establishing common notation and defining our notions of approximation in Section II. Section III presents algorithms and lower bounds for RLE. The algorithmic results are summarized in Theorem 1 and the lower bounds, in Theorem 2. Section IV deals with the LZ scheme: it starts with the structural lemmas, explains the approximation algorithm for compressibility with respect to LZ and finishes with the reduction from DE to LZ compressibility. Subsection IV-C describes a simple algorithm for DE.

II. Preliminaries

The input to our algorithms is a string w of length n over a finite alphabet Σ. Let $C(w)$ denote the length of the compressed version of w according to some fixed compression scheme.

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

We consider estimates to $C(w)$ that have both multiplicative and additive error. We call $\widehat{C}$ an (λ, ϵ)-*estimate* for $C(w)$ if

$$\frac{C(w)}{\lambda} - \epsilon n \;\leq\; \widehat{C} \;\leq\; \lambda \cdot C(w) + \epsilon n \ ,$$

and say an algorithm (λ, ϵ)-*estimates* C (or is an (λ, ϵ)-*approximation algorithm* for C) if for each input w it produces an (λ, ϵ)-estimate for $C(w)$ 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 $\lambda \geq \frac{1}{2\epsilon}$.

III. 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 $C_{\text{rle}}(w)$, is the sum over all runs of $\lceil \log(\ell + 1) \rceil + \lceil \log |\Sigma| \rceil$.

We assume that the alphabet Σ has constant size. This is a natural assumption when using run-length encoding, but the analysis of our algorithms can be extended in a straightforward manner to alphabets whose size is a function of n. The complexity of the algorithms will grow polylogarithmically with $|\Sigma|$.

We first present an algorithm that, given a parameter ϵ, outputs an ϵn-additive estimate to $C_{\text{rle}}(w)$ with high probability and makes $\tilde{O}(1/\epsilon^3)$ queries. We then reduce the query complexity to $\tilde{O}(1/\epsilon)$ at the cost of incurring a multiplicative approximation error in addition to additive: the new algorithm $(3, \epsilon)$-estimates $C_{\text{rle}}(w)$. We later discuss how to use approximation schemes with multiplicative and additive error to get a purely multiplicative approximation, at a cost on the query complexity that depends on $n/C_{\text{rle}}(w)$. That is, the more compressible the string w is, the higher the query complexity of the algorithm. These results are summarized in Theorem 1, stated next. The algorithms referred to by the theorem are presented in Subsections III-A–III-C.

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

1) *Algorithm I gives an ϵn-additive approximation to $C_{\text{rle}}(w)$ in time $\tilde{O}(1/\epsilon^3)$.*
2) *Algorithm II $(3, \epsilon)$-estimates $C_{\text{rle}}(w)$ in time $\tilde{O}(1/\epsilon)$.*

3) *Algorithm III* 4*-estimates* $C_{\mathrm{rle}}(w)$ *and runs 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* $C_{\mathrm{rle}}(w)$*.*

We also 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.*

In the next subsections we prove Theorems 1 and 2.

A. *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, $\frac{C_{\mathrm{rle}}(w)}{n} = \mathop{\mathsf{E}}_{t\in[n]}[c(t)]$, 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 $\epsilon n$-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 $\widehat{C}_{\mathrm{rle}} = n \cdot \mathop{\mathsf{E}}_{i\in[q]}[\hat{c}(t_i)]$.
```

Proof of Theorem 1, Item 1: We first prove that the algorithm is an ϵn-additive approximation algorithm. 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,

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

Therefore,

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

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

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, $\widehat{C}_{\rm rle}$ is an ϵn-additive estimate of $C_{\rm rle}(w)$, as desired.

We now turn to the query complexity an running time, where recall that we assume that $|\Sigma|$ 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 $O(\log(1/\epsilon)/\epsilon^3)$. ■

B. A $(3,\epsilon)$-Estimate with $\tilde{O}(1/\epsilon)$ Queries

If we are willing to allow a constant multiplicative approximation error in addition to ϵn-additive, we can reduce the query and time complexity to $\tilde{O}(1/\epsilon)$. The idea is to partition the positions in the string into *buckets* according to the length of the runs they belong to. Each bucket corresponds to runs of the same length up to a small constant factor. For the sake of brevity of the analysis, we take this constant to be 2. A smaller constant results in a better multiplicative factor. Given the definition of the buckets, for every two positions t_1 and t_2 from the same bucket, $c(t_1)$ and $c(t_2)$ differ by at most a factor of 2. Hence, good estimates of the sizes of all buckets would yield a good estimate of the total cost of the run-length encoding.

The algorithm and its analysis build on two additional observations: (1) Since the cost, $c(t)$, monotonically decreases with the length of the run to which t belongs, we can allow a less precise approximation of the size of the buckets that correspond to longer runs. (2) A bucket containing relatively few positions contributes little to the run-length encoding cost. Details follow.

```
ALGORITHM II: A $(3,\epsilon)$-APPROXIMATION FOR $C_{\rm rle}(w)$

1) Select $q = \Theta\left(\frac{\log(1/\epsilon)\cdot\log\log(1/\epsilon)}{\epsilon}\right)$ indices $t_1,\dots,t_q$ uniformly and independently at random.
2) For $h = 1,\dots,h_0$ do:
    a) Consider the first $q_h = \min\left\{q, q\cdot\frac{h+s}{2^{h-1}}\right\}$ indices $t_1,\dots,t_{q_h}$.
    b) For each $i = 1,\dots,q_h$, set $X_{h,i} = 1$ if $t_i \in B_h$ and set $X_{h,i} = 0$ otherwise.
3) Output $\widehat{C}_{\rm rle} = \sum_{h=1}^{h_0}\left(\frac{n}{q_h}\cdot\sum_{i=1}^{q_h} X_{h,i}\right)\cdot\frac{h+s}{2^{h-1}}$.
```

Proof of Theorem 1, Item 2: Let ℓ_0 be as defined in the previous subsection, and let $h_0 = \lceil \log \ell_0 \rceil$. Thus, $h_0 = O(\log(1/\epsilon))$. For each $h \in [h_0]$, let $B_h = \{t : 2^{h-1} \le \ell(t) < 2^h\}$.

That is, the bucket B_h contains all indices t that belong to runs of length approximately 2^h. Let $s \overset{\text{def}}{=} \lceil \log|\Sigma| \rceil$ and

$$C_{\text{rle}}(w,h) \overset{\text{def}}{=} \sum_{t \in B_h} c(t). \tag{4}$$

Then

$$|B_h| \cdot \frac{h+s}{2^h} \leq C_{\text{rle}}(w,h) \leq |B_h| \cdot \frac{h+s}{2^{h-1}}\ , \tag{5}$$

which implies that

$$C_{\text{rle}}(w,h) \leq |B_h| \cdot \frac{h+s}{2^{h-1}} \leq 2 \cdot C_{\text{rle}}(w,h)\ . \tag{6}$$

Our goal is to obtain (with high probability), for every h, a relatively accurate estimate β_h of $\frac{|B_h|}{n}$. Specifically, let

$$H_{\text{big}} = \left\{ h \ : \ \frac{|B_h|}{n} \geq \frac{1}{2} \cdot \frac{\epsilon}{h_0} \cdot \frac{2^{h-1}}{h+s} \right\} \quad \text{and} \quad H_{\text{small}} = \left\{ h \ : \ \frac{|B_h|}{n} < \frac{1}{2} \cdot \frac{\epsilon}{h_0} \cdot \frac{2^{h-1}}{h+s} \right\}. \tag{7}$$

Then we would like β_h to satisfy the following:

$$\begin{aligned} \frac{1}{3} \cdot \frac{|B_h|}{n} \leq \ & \beta_h \leq \tfrac{3}{2} \cdot \tfrac{|B_h|}{n} && \text{if } h \in H_{\text{big}}; \\ 0 \leq \ & \beta_h \leq \tfrac{\epsilon}{h_0} \cdot \tfrac{2^{h-1}}{h+s} && \text{otherwise } (h \in H_{\text{small}}). \end{aligned} \tag{8}$$

Given such estimates $\beta_1, \ldots, \beta_{h_0}$, approximate the encoding cost by $\widehat{C}_{\text{rle}} = \sum_{h=1}^{h_0} \beta_h \cdot n \cdot \frac{h+s}{2^{h-1}}$. Then

$$\begin{aligned} \widehat{C}_{\text{rle}} &= \sum_{h \in H_{\text{big}}} \beta_h \cdot n \cdot \frac{h+s}{2^{h-1}} + \sum_{h \in H_{\text{small}}} \beta_h \cdot n \cdot \frac{h+s}{2^{h-1}} && (9) \\ &\leq \sum_{h \in H_{\text{big}}} \frac{3}{2} \cdot |B_h| \cdot \frac{h+s}{2^{h-1}} + h_0 \cdot \frac{\epsilon}{h_0} \cdot \frac{2^{h-1}}{h+s} \cdot n \cdot \frac{h+s}{2^{h-1}} && (10) \\ &\leq \sum_{h \in H_{\text{big}}} 3 \cdot C_{\text{rle}}(w,h) + \epsilon n \ < \ 3 \cdot C_{\text{rle}}(w) + \epsilon n. && (11) \end{aligned}$$

The last inequality uses the upper bound from Equation (6). Similarly,

$$\begin{aligned} \widehat{C}_{\text{rle}} &\geq \sum_{h \in H_{\text{big}}} \beta_h \cdot n \cdot \frac{h+s}{2^{h-1}} && (12) \\ &\geq \frac{1}{3} \cdot \sum_{h \in H_{\text{big}}} C_{\text{rle}}(w,h) && (13) \\ &= \frac{1}{3} \cdot \left(C_{\text{rle}}(w) - \sum_{h \in H_{\text{small}}} C_{\text{rle}}(w,h) \right) && (14) \\ &> \frac{1}{3} \cdot C_{\text{rle}}(w) - \epsilon n && (15) \end{aligned}$$

Let β_h be a random variable equal to $\frac{1}{q_h}\sum_{i=1}^{q_h} X_{h,i}$. We show that with high probability, β_h satisfies Equation (8) for every $h \in [h_0]$. For each fixed h we have that $\Pr[X_{h,i}=1] = \frac{|B_h|}{n}$ for every $i \in [q_h]$. Hence, by a multiplicative Chernoff bound,

$$\Pr\left[\left|\beta_h - \frac{|B_h|}{n}\right| \geq \frac{1}{2}\frac{|B_h|}{n}\right] < \exp\left(-c \cdot \frac{|B_h|}{n} \cdot q_h\right) \tag{16}$$

for some constant $c \in (0,1)$. Recall that $h_0 = O(\log(1/\epsilon))$ and that $q_h = \Theta(q \cdot \frac{h+s}{2^{h-1}}) = \Omega\big(\epsilon^{-1} \cdot h_0 \cdot \log(h_0) \cdot \frac{h+s}{2^{h-1}}\big)$. Hence, for $h \in H_{\rm big}$ (and for a sufficiently large constant in the $\Theta(\cdot)$ notation in the definition of q), the probability in Equation (16) is at most $\frac{1}{3} \cdot \frac{1}{h_0}$, and so Equation (8) holds with probability at least $1 - \frac{1}{3} \cdot \frac{1}{h_0}$. On the other hand, for $h \in H_{\rm small}$, the probability that $\beta_h \geq \frac{\epsilon}{h_0} \cdot \frac{2^{h-1}}{h+s}$ is bounded above by the probability of this event when $\frac{|B_h|}{n} = \frac{1}{2} \cdot \frac{\epsilon}{h_0} \cdot \frac{2^{h-1}}{h+s}$. By Equation (16) this is at most $\frac{1}{3} \cdot \frac{1}{h_0}$, and so in this case too Equation (8) holds with probability at least $1 - \frac{1}{3} \cdot \frac{1}{h_0}$. By taking a union bound over all $h \in [h_0]$ the analysis is completed.

We now turn to the query complexity and running time. For a given index t_i, deciding whether $t_i \in B_h$ requires $O(2^h)$ queries. (More precisely, we need at most 2^{h-1} queries in addition to the queries from the previous iterations.) Hence, the total number of queries is

$$O\left(\sum_{h=1}^{h_0} q_h \cdot 2^h\right) = O\left(q \cdot h_0^2\right) = O\left(\frac{\log^3(1/\epsilon) \cdot \log\log(1/\epsilon)}{\epsilon}\right). \tag{17}$$

■

C. A 4-Multiplicative Estimate with $\tilde{O}(n/C_{\rm rle}(w))$ Queries

In this subsection we "get-rid" of the ϵn additive error by introducing a dependence on the run-length encoding cost (which is of course unknown to the algorithm). First, assume a lower bound $C_{\rm rle}(w) \geq \mu n$ for some $\mu > 0$. Then, by running Algorithm II (the $(3,\epsilon)$-approximation algorithm) with ϵ set to $\mu/2$, and outputting $\widehat{C}_{\rm rle} + \epsilon n$, we get a 4-multiplicative estimate with $\tilde{O}(1/\mu)$ queries.

We can search for such a lower bound μn, as follows. Suppose that Algorithm II receives, in addition to the additive approximation parameter ϵ, a confidence parameter δ, and outputs a $(3,\epsilon)$-estimate with probability at least $1-\delta$ instead of $2/3$. This can easily be achieved by increasing the query complexity of the algorithm by a factor of $\log(1/\delta)$. By performing calls to Algorithm II with decreasing values of ϵ and δ, we can maintain a sequence of intervals of decreasing size, that contain $C_{\rm rle}(w)$ (with high probability). Once the ratio between the extreme points of the interval is sufficiently small, the algorithm terminates. Details follow.

ALGORITHM III: A 4-APPROXIMATION FOR $C_{\rm rle}(w)$

```
1) Set $j = 0$, $lb_0 = 0$ and $ub_0 = 1$.
2) While $\frac{ub_j}{lb_j} > 16$ do:
    a) $j = j+1$, $\epsilon_j = 2^{-j}$, $\delta_j = \frac{1}{3} \cdot 2^{-j}$.
    b) Call Algorithm II with $\epsilon = \epsilon_j$ and $\delta = \delta_j$, and let $\widehat{C}^j_{\rm rle}$ be its output.
    c) Let $ub_j = 3(\widehat{C}^j_{\rm rle} + \epsilon_j n)$ and $lb_j = \frac{1}{3}(\widehat{C}^j_{\rm rle} - \epsilon_j n)$.
3) Output $\sqrt{lb_j \cdot ub_j}$.
```

Proof of Theorem 1, Item 3: For any given j, Algorithm II outputs $\widehat{C}^j_{\text{rle}} \in [\frac{1}{3}C_{\text{rle}}(w) - \epsilon_j n,\ 3C_{\text{rle}}(w) + \epsilon_j n]$, with probability at least $1 - \frac{1}{3} \cdot \delta_j$. Equivalently, $lb_j \leq C_{\text{rle}}(w) \leq ub_j$. By the Union bound, with probability at least $2/3$, $lb_j \leq C_{\text{rle}}(w) \leq ub_j$ for all j. Assume this event in fact holds. Then, upon termination (when $ub_j/lb_j \leq 16$), the output is a 4-multiplicative estimate of $C_{\text{rle}}(w)$. It is not hard to verify that once $\epsilon_j \leq \frac{C_{\text{rle}}(w)}{24n}$, then the algorithm indeed terminates with probability at least $1 - \delta_j$.

The query complexity of the algorithm is dominated by its last iteration. As stated above, for each $\epsilon_j \leq \frac{C_{\text{rle}}(w)}{24n}$, conditioned on the algorithm not terminating in iteration $j-1$, the probability that it does not terminate in iteration j is at most $\delta_j = \frac{1}{3}2^{-j}$. Since the query complexity of Algorithm II is $\tilde{O}(1/\epsilon)$, the expected query complexity of Algorithm III is $\tilde{O}(n/C_{\text{rle}}(w))$. ■

c) Improving the multiplicative approximation factor: The 4-multiplicative estimate of $C_{\text{rle}}(w)$ can be improved to a $(1+\gamma)$-multiplicative estimate for any $\gamma > 0$. This is done by refining the buckets defined in Subsection III-B so that $B_h = \{t : (1+\frac{\gamma}{2})^{h-1} \leq \ell(t) < (1+\frac{\gamma}{2})^h\}$ for $h = 1, \ldots, \log_{1+\frac{\gamma}{2}} \ell_0$ (=$O(\log(1/\epsilon)/\gamma)$), and setting $\epsilon = \gamma \cdot \mu/8$. The query complexity remains linear in $1/\mu = n/C_{\text{rle}}(w)$ (up to polylogarithmic factors), and is polynomial in $1/\gamma$.

D. A Multiplicative Lower Bound

The proof of Theorem 2, Item 1, follows from the next lemma, where we set $k = C$ and $k' = A^2 C \log n$.

Lemma 1. *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) $C_{\text{rle}}(w) = \Theta\left(k \log(\frac{n}{k})\right)$ 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, $C_{\text{rle}}(w) = \Theta\left(k \log(\frac{n}{k})\right)$. 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. ■

E. An Additive Lower Bound

Proof of 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.[5]

Let W be a random variable drawn according to $\mathcal{D}_{n,1/2}$ and W', according to $\mathcal{D}_{n,1/2+\epsilon}$. It is well known that $\Omega(1/\epsilon^2)$ independent coin flips are necessary to distinguish a coin with bias $\frac{1}{2}$ from a coin with bias $\frac{1}{2}+\epsilon$. Therefore, $\Omega(1/\epsilon^2)$ queries are necessary to distinguish w from w'.

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

We next show that with very high probability the encoding costs of w and w' differ by $\Omega(\epsilon n)$. Runs of length 1 contribute 1 to the encoding cost, and runs of length 3 cost $\lceil \log(3+1) \rceil = 2$. Therefore, each "heads" contributes $3 \cdot 1$, while each "tails" contributes 2. Hence, if we got $\alpha \cdot \frac{n}{3}$ "heads", then the encoding cost of the resulting string is $\frac{n}{3} \cdot (3\alpha + 2(1-\alpha)) = \frac{n}{3} \cdot (2+\alpha)$. The expected value of α is p. By an additive Chernoff bound, $|\alpha - p| \le \epsilon/4$ with probability at least $1 - 2\exp(-2(\epsilon/4)^2)$. With this probability, the encoding cost of the selected string is between $\frac{n}{3} \cdot \left(2 + p - \frac{\epsilon}{4}\right)$ and $\frac{n}{3} \cdot \left(2 + p + \frac{\epsilon}{4}\right)$. The theorem (for the case $n \mod 3 = 0$) follows, since with very high probability, $C_{\text{rle}}(w') - C_{\text{rle}}(w) = \Omega(\epsilon n)$.

If $n \mod 3 = b$ for some $b > 0$ then we make the following minor changes in the construction and the analysis: (1) The first b bits in the string are always set to 0. (2) This adds b to the encoding cost. (3) Every appearance of $\frac{n}{3}$ in the proof is replaced by $\left\lfloor \frac{n}{3} \right\rfloor$. It is easy to verify that the lower bound holds for any sufficiently large n. ■

IV. Lempel Ziv Compression

In this section we consider a variant of Lempel and Ziv's compression algorithm [4], which we refer to as LZ77. In all that follows we use the shorthand $[n]$ for $\{1, \dots, n\}$. 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 $w_t \dots w_{t+\ell-1}$ for which there exists an index $p < t$, such that $w_p \dots w_{p+\ell-1} = w_t \dots w_{t+\ell-1}$. (The substrings $w_p \dots w_{p+\ell-1}$ and $w_t \dots w_{t+\ell-1}$ 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 $w_t \dots w_{t+\ell-1}$ (or w_t when w_t is a new character) as a *compressed segment*. Clearly, compression takes time $O(n^2)$, and decompression, time $O(n)$.

Let $C_{\text{LZ}}(w)$ 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 $\widehat{C}_{\text{LZ}}$ of $C_{\text{LZ}}(w)$. 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 symbols 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 $C_{\text{LZ}}(w)$. This is relatively negligible given the quality of the estimates that we can achieve in sublinear time.

Our results on approximating LZ compressibility can be summarized succinctly:

Theorem 3. *For any alphabet Σ:*

1) *Algorithm IV (A, ϵ)-estimates $C_{\text{LZ}}(w)$ and runs in time $\tilde{O}\left(\frac{n}{A^3 \epsilon}\right)$.*
2) *For any $B = n^{o(1)}$, distinguishing strings with LZ compression cost $\tilde{\Omega}(n)$ from strings with cost $\tilde{O}(n/B)$ requires $n^{1-o(1)}$ queries.*

The first bound states that non-trivial approximation guarantees are indeed possible. For example, by setting $A = o(n^{\alpha/2})$ and $\epsilon = o(n^{-\alpha/2}$, we get an algorithm which distinguishes incompressible strings ($C_{\text{LZ}} = \tilde{\Omega}(n)$) from partly compressible strings ($C_{\text{LZ}} = O(n^{\alpha})$) in sublinear time $\tilde{O}(n^{1-\alpha})$. The lower bound states that in some sense this is tight: no approximation algorithm with a purely additive approximation guarantee can run in time which is significantly sublinear.

In the remainder of this section we develop the tools necessary to prove the theorem. We begin by relating LZ compressibility to DE (Section IV-A), then use this relation to discuss algorithms (Section IV-B) and lower bounds (Section IV-D) for compressibility.

A. Structural Lemmas

Our algorithm for approximating the compressibility of an input string with respect to LZ77 uses an approximation algorithm for DE (defined in the introduction) as a subroutine. The main tool in the reduction from LZ77 to DE is the relation between $C_{\rm LZ}(w)$ 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 2 (Structural Lemma 1)**.** *For every $\ell \in [n]$, $C_{LZ}(w) \geq \frac{d_\ell(w)}{\ell}$.*

Lemma 3 (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 2: This proof is similar to the proof of a related lemma concerning grammars from [17]. 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 length 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, $d_\ell(w) \leq (C_{\rm LZ}(w) - 1)(\ell - 1) < C_{\rm LZ}(w) \cdot \ell$ for every $\ell > 1$. ■

Proof of Lemma 3: 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 $1 \leq \ell \leq \lfloor \ell_0/2 \rfloor$,

$$\sum_{k=1}^{\ell} n_k \leq 2(m+1) \cdot \sum_{k=1}^{\ell} \frac{1}{k} \; . \tag{18}$$

For all $\ell \geq 1$, since the compressed segments in w are disjoint,

$$\sum_{k=\ell+1}^{n} n_k \leq \frac{n}{\ell+1} \; . \tag{19}$$

If we substitute $\ell = \lfloor \ell_0/2 \rfloor$ in Equations (18) and (19), and sum the two equations, we get:

$$\sum_{k=1}^{n} n_k \leq 2(m+1) \cdot \sum_{k=1}^{\lfloor \ell_0/2 \rfloor} \frac{1}{k} + \frac{2n}{\ell_0} \leq 2(m+1)(\ln \ell_0 + 1) + \frac{2n}{\ell_0}. \tag{20}$$

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

It remains to prove Equation (18). We do so below by induction on ℓ, using the following claim.

Claim 4. *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 $j \in \{\ell, \dots, n-\ell\}$ 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, $w_t \dots w_{t+\ell-1}$ is *new* if there is no $p < t$ such that $w_t \dots w_{t+\ell-1} = w_p \dots w_{p+\ell-1}$. Consider a compressed substring $w_t \dots w_{t+k-1}$ 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 $w_{t'} \dots w_{t+k'}$ where $t' \leq t$ and $k' \geq k + (t'-t)$, is *new*.

Map each position $j \in \{\ell, \dots, n-\ell\}$ in the compressed substring $w_t \dots w_{t+k-1}$ to the length-2ℓ substring that ends at $w_{j+\ell}$. Then each position in $\{\ell, \dots, n-\ell\}$ that appears in a compressed substring of length at most ℓ is mapped to a distinct length-2ℓ substring, as desired. ■
(Claim 4)

d) Establishing Equation (18): We prove Equation (18) by induction on ℓ. Claim 4 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 4 to the sum of the induction hypothesis inequalities (Equation (18)) 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

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

Dividing both sides by ℓ gives the inequality in Equation (18). ■

e) Tightness of Lemma 3: The following lemma shows that Lemma 3 is asymptotically tight.

Lemma 5. *For all positive integers m and $\ell_0 \leq m$, there is a string w of length n ($n \approx m(\ell_0 + \ln \ell_0)$) with $O(\ell m)$ distinct substrings of length ℓ for each $\ell \in [\ell_0]$, such that $C_{LZ}(w) = \Omega(m \log \ell_0 + n/\ell_0)$.*

Proof: We construct such *bad* strings over the alphabet $[m]$. A *bad* string is constructed in ℓ_0 phases, where in each new phase, ℓ, we add a substring of length between m and $2m$ that might repeat substrings of length up to ℓ that appeared in the previous phases, but does not repeat longer substrings. Phase 1 contributes the string '$1 \ldots m$'. In phase $\ell > 1$, we list characters 1 to m in the increasing order, repeating all characters divisible by $\ell - 1$ twice. For example, phase 2 contributes the string '11 22 33$\ldots mm$', phase 3 the string '122 344 566$\ldots m$', phase 4 the string '1233 4566 7899$\ldots m$', etc. The spaces in the strings are introduced for clarity.

First observe that the length of the string, n, is at most $2m\ell_0$. Next, let us calculate the number of distinct substrings of various sizes. Since the alphabet size is m, there are m length-1 substrings. There are at most $2m$ length-2 substrings: 'i i' and 'i $(i+1)$' for every i in $[m-1]$, as well as 'm m' and 'm 1'. We claim that for $1 < \ell \leq \ell_0$, there are at most $3\ell m$ length-ℓ substrings. Specifically, for every i in $[m]$, there are at most 3ℓ length-ℓ substrings that start with i. This is because each of the first ℓ phases contributes at most 2 such substrings: one that starts with 'i $(i+1)$', and one that starts with 'i i'. In the remaining phases a length-ℓ substring can have at most one repeated character, and so there are ℓ such substrings that start with i. Thus, there are at most $\ell \cdot 3m$ distinct length-ℓ substrings in the constructed string.

Finally, let us look at the cost of LZ77 compression. It is not hard to see that ℓth phase substring compresses by at most a factor of ℓ. Since each phase introduces a substring of length at least m, the total compressed length is at least $m(1 + 1/2 + 1/3 + \ldots + 1/\ell_0) = \Omega(m \log \ell_0) = \Omega(m \log \ell_0 + n/\ell_0)$. The last equality holds because $n \leq 2m\ell_0$ and, consequently, $\frac{n}{\ell_0} = o(m \log \ell_0)$. ■

In the proof of Lemma 5 the alphabet size is large. It can be verified that by replacing each symbol from the large alphabet $[m]$ with its binary representation, we obtain a binary string of length $\Theta(m \log m\ell_0)$ with the properties stated in the lemma.

B. 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 DE (Definition 1) as a subroutine. The main tool in the reduction from LZ77 to DE consists of structural lemmas 2 and 3, summarized in the following corollary.

Corollary 4. *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 for DE, described in Subsection IV-C, as a subroutine. Recall that an algorithm for DE approximates the number of distinct symbols in an input string. We denote the number of distinct symbols in an input string τ by $C_{\text{DSS}}(\tau)$. To approximate d_ℓ, the number of distinct length-ℓ substrings in w, using an algorithm for DE, view each length-ℓ substring as a separate symbol. Each query of the algorithm for DE can be implemented by ℓ queries to w.

Let $\textsc{Estimate}(\ell, B, \delta)$ 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 DE, 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. By Lemma 7, the query complexity of $\textsc{Estimate}(\ell, B, \delta)$ is $O\left(\frac{n}{B^2}\, \ell \log \delta^{-1}\right)$. Using $\textsc{Estimate}(\ell, B, \delta)$ as a subroutine, we get the following approximation algorithm for the cost of LZ77.

```
ALGORITHM IV: AN $(A, \epsilon)$-APPROXIMATION FOR $C_{\text{LZ}}(w)$
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 $\hat{d}_\ell = \textsc{Estimate}(\ell, B, \frac{1}{3\ell_0})$.
3) Combine the estimates to get an approximation of $m$ from Corollary 4:   set $\hat{m} = \max_\ell \frac{\hat{d}_\ell}{\ell}$.
4) Output $\widehat{C}_{\text{LZ}} = \hat{m} \cdot \frac{A}{B} + \epsilon n$.
```

Lemma 6 (Theorem 3, part 1, restated)**.** *Algorithm IV (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 4, which implies that

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

Equivalently, $\dfrac{C_{\text{LZ}} - 4(n/\ell_0)}{4B \log \ell_0} \le \hat{m} \le B \cdot C_{\text{LZ}}$. 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 $\widehat{C}_{\text{LZ}}$ 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. ■

C. *A Simple Algorithm for* DE

Here we describe a simple approximation algorithm for DE. The Guaranteed-Error estimator of Charikar *et al.* has the same guarantees as our approximation algorithm. Our algorithm is (even) simpler, and we present it here for completeness.

Algorithm V: A λ-approximation for DE

1) Take $\frac{10n}{\lambda^2}$ samples from the string τ.
2) Let $\widehat{C}$ be the number of distinct symbols in the sample; output $\widehat{C} \cdot \lambda$.

Lemma 7. *Let $\lambda = \lambda(n)$. Algorithm V is an λ-approximation algorithm for* DE *whose query complexity and running time are $O\left(\frac{n}{\lambda^2}\right)$.*

Proof: Let C be the number of distinct symbols in the string τ. We need to show that $\frac{C}{\lambda} \le \widehat{C} \cdot \lambda \le C \cdot \lambda$, or equivalently, $\frac{C}{\lambda^2} \le \widehat{C} \le C$, with probability at least $\frac{2}{3}$. The sample always contains at most as many distinct symbols as there are in τ: $\widehat{C} \le C$. Claim 8, stated below and applied with $s = \frac{10n}{\lambda^2}$, shows that $\widehat{C} \ge \frac{C}{\lambda^2}$ with probability $\ge \frac{2}{3}$. To get the running time $O\left(\frac{n}{\lambda^2}\right)$ one can use a random 2-universal hash function. ■

Claim 8. *Let $s = s(n) \le n$. Then s independent samples from a distribution with $C = C(n)$ elements, where each element has probability $\ge \frac{1}{n}$, yield at least $\frac{Cs}{10n}$ distinct elements, with probability $\ge \frac{3}{4}$.*

Proof: For $i \in [C]$, let X_i be the indicator variable for the event that color i is selected in s samples. Then $X = \sum_{i=1}^{C} X_i$ is a random variable for the number of distinct colors. Since each color is selected with probability at least $\frac{1}{n}$ for each sample,

$$\mathsf{E}[X] = \sum_{i=1}^{C} \mathsf{E}[X_i] \ge C\left(1 - \left(1 - \frac{1}{n}\right)^s\right) \ge C\left(1 - e^{-(s/n)}\right) \ge (1 - e^{-1})\frac{Cs}{n}. \tag{21}$$

The last inequality holds because $1 - e^{-x} \geq (1 - e^{-1}) \cdot x$ for all $x \in [0, 1]$.

We now use Chebyshev's inequality to bound the probability that X is far from its expectation. For any distinct pair of colors i, j, the covariance $\mathsf{E}(X_i X_j) - \mathsf{E}(X_i)\,\mathsf{E}(X_j)$ is negative (knowing that one color was not selected makes it more likely for any other color to be selected). Since X is a sum of Bernoulli variables, $\mathsf{Var}[X] \leq \mathsf{E}[X]$. For any $\delta > 0$,

$$\Pr\left[X \leq \delta\,\mathsf{E}[X]\right] \leq \Pr\left[|X - \mathsf{E}[X]| \geq (1-\delta)\,\mathsf{E}[X]\right] \leq \frac{\mathsf{Var}[X]}{((1-\delta)\,\mathsf{E}[X])^2} \leq \frac{1}{(1-\delta)^2\,\mathsf{E}[X]}. \quad (22)$$

Set $\delta = 3 - \sqrt{8}$. If $\mathsf{E}[X] \geq \frac{4}{(1-\delta)^2}$, then by Equations (22) and (21), with probability $\geq \frac{3}{4}$, variable $X \geq \delta\,\mathsf{E}[X] \geq \delta(1 - e^{-1})\frac{Cs}{n} > \frac{Cs}{10n}$, as stated in the claim. Otherwise, that is, if $\mathsf{E}[X] < \frac{4}{(1-\delta)^2}$, Equation (21) implies that $\frac{4\delta}{(1-\delta)^2} > \delta(1-e^{-1})\frac{Cs}{n}$. Substituting $3 - \sqrt{8}$ for δ gives $1 > \frac{Cs}{10n}$. In other words, the claim for this case is that at least one color appears among the samples, which, clearly, always holds. ■

D. Lower Bounds: Reducing DE *to LZ77*

We have demonstrated that estimating the LZ77 compressibility of a string reduces to DE. As shown in [12], DE is quite hard, and it is not possible to improve much on the simple approximation algorithm in Subsection IV-C, 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 DE? As we shall see, it is not much easier in general.

Lemma 9 (Reduction from DE 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 $C_{LZ}(w) \leq \alpha n$ and the case that $C_{LZ}(w) > \beta n$, for some $\alpha < \beta$.*

Then there is an algorithm for DE *taking inputs of length $n' = \Theta(\alpha n)$ that performs q queries and, with probability at least $2/3$, distinguishes inputs with at most $\alpha' n'$ distinct symbols from those with at least $\beta' n'$ distinct symbols, $\alpha' = \alpha/2$ 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 DE 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 DE 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. [12] give a strong lower bound on the sample complexity of approximation algorithms for DE. An interesting special case is that a subpolynomial-factor approximation for DE requires many queries even with a promise that the strings are only slightly compressible: for any $B = n^{o(1)}$, distinguishing inputs with $n/11$ distinct symbols from those with n/B distinct symbols requires $n^{1-o(1)}$ queries. Lemma 9 extends that bound to estimating LZ compressibility, as stated in Theorem 3. In fact, the lower bound for DE in [12] applies to a broad range of parameters, and yields the following general statement when combined with Lemma 9:

Corollary 5 (LZ is Hard to Approximate with Few Samples). *For sufficiently large n, all alphabets Σ and all $B \leq n^{1/4}/(4\log n^{3/2})$, 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 $C_{LZ}(w) \leq \alpha n$ 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 $\mathcal{A}_{\mathrm{LZ}}$ for LZ-compressibility as specified in the premise of Lemma 9. Here we show how to transform a DE instance τ into an input for $\mathcal{A}_{\mathrm{LZ}}$, and use the output of $\mathcal{A}_{\mathrm{LZ}}$ to distinguish τ with at most $\alpha' n'$ distinct symbols from τ with at least $\beta' n'$ distinct symbols, 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 DE, we transformed substrings into single symbols. Here we perform the reverse operation.

Given a DE 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 $\mathcal{A}_{\mathrm{LZ}}$ on w to obtain information about τ. We begin by replacing each distinct symbol 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 $\alpha' n'$ distinct symbols, then $C_{\mathrm{LZ}}(w) \leq 2\alpha' n$;
2) If τ has at least $\beta' n'$ distinct symbols, then $\Pr_w[C_{\mathrm{LZ}}(w) \geq \frac{1}{2} \cdot \min\left\{1, \frac{\log|\Sigma|}{4\log n'}\right\} \cdot \beta' n] \geq \frac{7}{8}$.

That is, in the first case we get an input w such that $C_{\mathrm{LZ}}(w) \leq \alpha n$ for $\alpha = 2\alpha'$, and in the second case, with probability at least $7/8$, $C_{\mathrm{LZ}}(w) \geq \beta n$ 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 $C_{\mathrm{DSS}}(\tau) \leq \alpha' n'$ from the case that $C_{\mathrm{DSS}}(\tau) > \beta' n'$, 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 symbol in τ that corresponds to the block. If this symbol was not yet observed, then we set the block to a uniformly selected substring in Σ^k. If this symbol was already observed in τ, then we set the block according to the substring that was already selected for the symbol. 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 $\alpha' n'$ distinct symbols 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,

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

If τ contains $\beta' n'$ or more distinct symbols, 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

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

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

Acknowledgments: We would like to thank Amir Shpilka, who was involved in a related paper on distribution support testing [12] 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] M. Li and P. Vitányi, *An Introduction to Kolmogorov Complexity and Its Applications*. Springer, 1997.

[2] T. Batu, S. Dasgupta, R. Kumar, and R. Rubinfeld, "The complexity of approximating the entropy." *SIAM Journal on Computing*, vol. 35, no. 1, pp. 132–150, 2005.

[3] M. Brautbar and A. Samorodnitsky, "Approximating the entropy of large alphabets," in *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, 2007.

[4] J. Ziv and A. Lempel, "A universal algorithm for sequential data compression," *IEEE Transactions on Information Theory*, vol. 23, pp. 337–343, 1977.

[5] T. Cover and J. Thomas, *Elements of Information Theory*. Wiley & Sons, 1991.

[6] D. Loewenstern, H. Hirsh, M. Noordewier, and P. Yianilos, "DNA sequence classification using compression-based induction," Rutgers University, DIMACS, Tech. Rep. 95-04, 1995.

[7] O. V. Kukushkina, A. A. Polikarpov, and D. V. Khmelev, "Using literal and grammatical statistics for authorship attribution," *Prob. Peredachi Inf.*, vol. 37, no. 2, pp. 96–98, 2000, [Probl. Inf. Transm. (Engl. Transl.) 37, 172–184 (2001)].

[8] D. Benedetto, E. Caglioti, and V. Loreto, "Language trees and zipping," *Phys. Rev. Lett.*, vol. 88, no. 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.

[9] M. Li, X. Chen, X. Li, B. Ma, and P. M. B. Vitányi, "The similarity metric." *IEEE Transactions on Information Theory*, vol. 50, no. 12, pp. 3250–3264, 2004, prelim. version in *SODA 2003*.

[10] R. Cilibrasi and P. M. B. Vitányi, "Clustering by compression." *IEEE Transactions on Information Theory*, vol. 51, no. 4, pp. 1523–1545, 2005.

[11] ——, "Similarity of objects and the meaning of words." in *TAMC*, ser. Lecture Notes in Computer Science, J. Cai, S. B. Cooper, and A. Li, Eds., vol. 3959. Springer, 2006, pp. 21–45.

[12] S. Raskhodnikova, D. Ron, A. Shpilka, and A. Smith, "Strong lower bounds for approximating distribution support size and the distinct elements problem," in *Proceedings of the Forty-Eight Annual Symposium on Foundations of Computer Science*, 2007, to appear.

[13] M. Charikar, S. Chaudhuri, R. Motwani, and V. R. Narasayya, "Towards estimation error guarantees for distinct values." in *PODS*. ACM, 2000, pp. 268–279.

[14] J. Bunge, "Bibligraphy on estimating the number of classes in a population," `www.stat.cornell.edu/~bunge/bibliography.htm`.

[15] N. Alon, Y. Matias, and M. Szegedy, "The space complexity of approximating the frequency moments." *J. Comput. Syst. Sci.*, vol. 58, no. 1, pp. 137–147, 1999.

[16] Z. Bar-Yossef, R. Kumar, and D. Sivakumar, “Sampling algorithms: lower bounds and applications,” in *Proceedings of the thirty-third annual ACM symposium on Theory of computing*. New York, NY, USA: ACM Press, 2001, pp. 266–275.

[17] E. Lehman and A. Shelat, “Approximation algorithms for grammer-based compression,” in *Proc. 18th Annual Symp. on Discrete Algorithms*, 2002, pp. 205–212.

[18] J. Ziv and A. Lempel, “Compression of individual sequences via variable-rate coding,” *IEEE Transactions on Information Theory*, vol. 24, pp. 530–536, 1978.