Polylog space compression is incomparable with Lempel-Ziv and pushdown compression

Elvira Mayordomo[1⋆] and Philippe Moser[2⋆]

[1] Departamento de Informática e Ingeniería de Sistemas, María de Luna 1, Universidad de Zaragoza, 50018 Zaragoza, SPAIN. elvira(at)unizar.es
[2] Department of Computer Science, National University of Ireland Maynooth, Maynooth, Co. Kildare, Ireland. pmoser(at)cs.nuim.ie

Abstract. This paper considers online compression algorithms that use at most polylogarithmic space (plogon). These algorithms correspond to compressors in the data stream model. We study the performance attained by these algorithms and show they are incomparable with both pushdown compressors and the Lempel-Ziv compression algorithm.

Keywords: compression algorithms, plogon, computational complexity, data stream algorithms, Lempel-Ziv algorithm, pushdown compression.

1 Introduction

The compression algorithms that are required for today massive data applications necessarily fall under very limited resource restrictions. In the case of the data stream setting, the algorithm sees a stream of elements one-by-one and does not have enough memory to store the whole stream, in fact the amount of available memory is far below linear [3, 12]. In the context of XML data bases the main limiting factor being document size renders the use of syntax directed compression particularly appropriate, i.e. compression centered on the grammar-based generation of XML-texts and performed with stack memory [9, 15].

In this paper we introduce the concept of polylogarithmic space online compressors (plogon), i.e., compression algorithms that use at most polylogarithmic memory while accessing the input only once. This type of algorithms models the compression that can actually be performed in the setting of data streams.

We compare the performance of plogon compressors with two important cases of lossless restricted memory compression algorithms, namely Lempel-Ziv and pushdown (stack memory) compressors.

The widely-used Lempel-Ziv algorithm LZ78 [16] was introduced as a general purpose compression algorithm that compresses any sequence at least as well as any finite-state compressors (in terms of asymptotic compression ratio). This means that for infinite sequences, the algorithm attains the (a posteriori) finite

⋆ Research supported in part by Spanish Government MEC and the European Regional Development Fund (ERDF) under Project TIN 2005-08832-C03-02.

state or block entropy. If we consider an ergodic source, the Lempel-Ziv compression coincides exactly with the entropy of the source with high probability on finite inputs. This second result is useful when the data source is known, but it is not informative for arbitrary inputs, i.e. when the data source is unknown (notice that an infinite sequence is Lempel-Ziv incompressible with probability one). Therefore for the comparison of compression algorithms on general sequences, either an experimental or a formal approach is needed, such as that used in [14]. In this paper we follow [14] using a worst case approach, that is, we consider asymptotic performance on every infinite sequence.

We shall use our formalization of pushdown compression (stack memory) from [2] and [1], together with a more feasible model where the pushdown compressor is required to be invertible by a pushdown transducer (see Section 3.2). As mentioned before, stack compression is especially adequate for XML-texts and has been extensively used [9, 15]. In fact XML compression can be performed with an even more restrictive computation model, known as visibly pushdown automata [4, 13], in which the input symbols are distinguished in three categories, namely call symbols, return symbols, and internal symbols, that determine the type of stack movement, that is, push, pop, or null.

In this paper we study the performance of plogon compressors compared to Lempel-Ziv's algorithm and pushdown compressors. This comparison is made in terms of asymptotic compression ratio for infinite sequences, and without any a priori assumption on the data's source.

We prove that the performance of plogon compressors and Lempel-Ziv's compression scheme is incomparable in the most general sense. We construct a sequence that Lempel-Ziv compresses optimally but that any plogon transducer fails to compress at all, and vice-versa. In both cases the separation is the strongest possible, i.e. optimal compressibility is achieved in the worst case (i.e. almost all prefixes of the sequence are optimally compressible), whereas incompressibility is present even in the best case (i.e. only finitely many prefixes of the sequence are compressible).

For the comparison of plogon and pushdown transducers, we use the more general pushdown model (where the pushdown compressor need not be invertible by a pushdown transducer) for incompressibility and the more restrictive (where the pushdown compressor is required to be invertible by a pushdown transducer) for compressibility, thus obtaining the most general results. We show that the more restrictive invertible pushdown compressors or the visibly pushdown ones achieve a worst case compression that no plogon compressor can attain, even in the best case. On the other hand plogon compressors obtain optimal compression on a sequence where pushdown compression is useless even with the help of endmarkers.

In [2] we studied the relation between pushdown and Lempel-Ziv compressibility, and showed their incomparability. This together with the results in this paper, shows the incomparability of the three restricted memory compression schemes that are Lempel-Ziv, pushdown compressors, and plogon compressors.

The paper is organized as follows. Section 2 contains some preliminaries. In section 3, we present our model of plogon compressor along with some basic properties and notations, as well as a review of both pushdown compressors and the Lempel-Ziv (LZ78) algorithm. In section 4 we show that plogon compression is incomparable with both Lempel-Ziv and pushdown compression. We end with a brief conclusion on connections and consequences of these results for effective dimension and prediction algorithms.

Due to lack of space, full proofs will appear in the journal version of this paper.

2 Preliminaries

Let us fix some notation for strings and languages. Let Σ be finite alphabet. A *string* is an element of Σ^n for some integer n and a *sequence* is an element of Σ^∞. For a string x, its length is denoted by $|x|$. If x, y are strings, we write $x \leq y$ (called lexicographic order) if $|x| < |y|$ or $|x| = |y|$ and x precedes y in alphabetical order. The empty string is denoted by λ. For $S \in \Sigma^\infty$ and $i, j \in \mathbb{N}$, we write $S[i..j]$ for the string consisting of the i^{th} through j^{th} bits of S, with the convention that $S[i..j] = \lambda$ if $i > j$, and $S[0]$ is the leftmost bit of S. We say string y is a prefix of string (sequence) x, denoted $y \sqsubset x$, if there exists a string (sequence) a such that $x = ya$. For a string x, x^{-1} denotes x written in reverse order, and $\bar{x}$ is the string obtained by flipping every bit of x (in the case x is a binary string). For a function $f : A \to B$, $f(x) = \perp$ means f is not defined on input x.

Given a sequence S and an (injective) function $T : \Sigma^* \to \Sigma^*$, the upper and lower compression ratios are given by

$$\rho_T(S) = \liminf_{n\to\infty} \frac{|T(S[1\ldots n])|}{n}, \text{ and}$$
$$R_T(S) = \limsup_{n\to\infty} \frac{|T(S[1\ldots n])|}{n}.$$

Given a sequence S and a class of functions $\mathcal{T}$, the upper and lower compression ratios are given by

$$\rho_{\mathcal{T}}(S) = \inf_{T\in\mathcal{T}} \rho_T(S), \text{ and}$$
$$R_{\mathcal{T}}(S) = \inf_{T\in\mathcal{T}} R_T(S).$$

$\rho_{\mathcal{T}}(S)$ corresponds to the best-case performance of $\mathcal{T}$-compressors on S and $R_{\mathcal{T}}(S)$ corresponds to the worst-case performance of $\mathcal{T}$-compressors on S.

Notation. We use $K(w)$ to denote the standard (plain) Kolmogorov complexity, that is, fix a universal Turing Machine U. Then for each string w,

$$K(w) = \min\{|p| \,|\, U(p) = w\}$$

i.e., $K(w)$ is the size of the shortest program that makes U output w. Although some authors use $C(w)$ to denote (plain) Kolmogorov complexity, we reserve this notation to denote a particular compression algorithm C on input w.

3 Compressors with low resource-bounds

In this section we review several families of lossless compression methods that use very low computing resources, in particular that of poly-logarithmic space computable compressors, stack-computable compressors, and the celebrated Lempel-Ziv algorithm.

3.1 plogon compressors

Let us define the family of compressors that can be computed online with at most poly-logarithmic space. Notice that these resource bounds match exactly those of the data stream model [3, 12], where the input size is massive in comparison with the available memory, and the input can only be read once.

Definition 3.1. *(Hartmanis, Immerman, Mahaney [10]) A Turing machine M is a plogon transducer if it has the following properties, for each input string w*

- *the computation of $M(w)$ reads its input from left to right (no turning back),*
- *$M(w)$ is given $|w|$ written in binary (on a special tape),*
- *$M(w)$ writes the output from left to right on a write-only output tape,*
- *$M(w)$ uses memory bounded by $\log(|w|)^c$, for a constant c.*

We denote with plogon the class of plogon transducers.

Note that contrary to Finite State transducers, a plogon transducer is not necessarily a mere extender, i.e., there is a plogon transducer M and strings w, x such that $M(wx) \not\supset M(w)$.

Definition 3.2. *A plogon transducer $C : \{0,1\}^* \to \{0,1\}^*$ is an information lossless compressor (ILpC) if it is 1-1.*

Notation. The best-case *plogon compression ratio* of a sequence $S \in \Sigma^\infty$ is $\rho_{\text{plogon}}(S) = \inf\{\rho_C(S) \mid \ C \text{ is an ILpC}\}$.
The worst-case *plogon compression ratio* of a sequence $S \in \Sigma^\infty$ is $R_{\text{plogon}}(S) = \inf\{R_C(S) \mid \ C \text{ is an ILpC}\}$.

3.2 Pushdown compressors

We next introduce the definition of pushdown computable compressors from [1]. It corresponds to the natural concept of pushdown computation of a single function that can be an information-lossless compressor.

The use of pushdown or stack compression has recently received new attention in the design of new compression schemes for XML where the use of syntax directed compression is particularly adequate [9, 15]. In the context of XML, a restriction of pushdown automata is used, known as visibly pushdown automata [4, 13], which are a restriction of the concept below. Therefore the compression ratio attained by visibly pushdown automata is an upper bound on the compression ratio attained through the pushdown compressors defined next.

Definition 3.3. *A bounded pushdown compressor (BPDC) is an 8-tuple*

$$C = (Q, \Sigma, \Gamma, \delta, \nu, q_0, z_0, c)$$

where

- Q *is a finite set of states*
- Σ *is the finite input alphabet*
- Γ *is the finite stack alphabet*
- $\delta : Q \times (\Sigma \cup \{\lambda\}) \times \Gamma \to Q \times \Gamma^*$ *is the transition function*
- $\nu : Q \times \Sigma \times \Gamma \to \Sigma^*$ *is the output function*
- $q_0 \in Q$ *is the initial state*
- $z_0 \in \Gamma$ *is the start stack symbol*
- $c \in \mathbb{N}$ *is an upper bound on the number of* λ*-rules per input symbol*

We use δ_Q and δ_{Γ^*} for the projections of function δ. Note that the transition function δ accepts λ as an input character in addition to elements of Σ, which means that C has the option of not reading an input character while altering the stack, such a movement is called a λ-*rule*. In this case $\delta(q, \lambda, a) = (q', \lambda)$, that is, we pop the top symbol of the stack. To enforce determinism, we require that at least one of the following hold for all $q \in Q$ and $a \in \Gamma$:

- $\delta(q, \lambda, a) = \perp$,
- $\delta(q, b, a) = \perp$ for all $b \in \Sigma$.

We restrict δ so that z_0 cannot be removed from the stack bottom, that is, for every $q \in Q$, $b \in \Sigma \cup \{\lambda\}$, either $\delta(q, b, z_0) = \perp$, or $\delta(q, b, z_0) = (q', vz_0)$, where $q' \in Q$ and $v \in \Gamma^*$.

There are several natural variants for the model of pushdown transducer [5], both allowing different degrees of nondeterminism and computing partial (multi)functions by requiring final state or empty stack termination conditions. Our purpose is to compute a total and well-defined (single valued) function in order to consider general-purpose, information-lossless compressors.

Notice that so far we have not required that the computation should be invertible by another pushdown transducer, which is a natural requirement for practical compression schemes. We shall also consider an invertible notion of pushdown compressors, where we explicitly require the computation to be invertible by another pushdown transducer; more details to follow at the end of the present section.

We first consider the transition function δ as having inputs in $Q \times (\Sigma \cup \{\lambda\}) \times \Gamma$, meaning that only the top symbol of the stack is relevant. Then we use the extended transition function $\delta^* : Q \times \Sigma^* \times \Gamma^+ \to Q \times \Gamma^*$, defined recursively as follows. For $q \in Q$, $v \in \Gamma^+$, $w \in \Sigma^*$, and $b \in \Sigma$

$$\delta^*(q, \lambda, v) = \begin{cases} \delta^*(\delta_Q(q, \lambda, v), \lambda, \delta_{\Gamma^*}(q, \lambda, v)), & \text{if } \delta(q, \lambda, v) \neq \perp; \\ (q, v), & \text{otherwise.} \end{cases}$$

$$\delta^*(q, wb, v) = \begin{cases} \delta^*(\delta_Q(\delta_Q^*(q, w, v), b, \delta_{\Gamma^*}^*(q, w, v)), \lambda, \delta_{\Gamma^*}(\delta_Q^*(q, w, v), b, \delta_{\Gamma^*}^*(q, w, v))), \\ \qquad \text{if } \delta^*(q, w, v) \neq \perp \text{ and } \delta(\delta_Q^*(q, w, v), b, \delta_{\Gamma^*}^*(q, w, v)) \neq \perp; \\ \perp, \quad \text{otherwise.} \end{cases}$$

That is, λ-rules are implicit in the definition of δ^*. We abbreviate δ^* to δ, and $\delta(q_0, w, z_0)$ to $\delta(w)$. We define the *output* from state q on input $w \in \Sigma^*$ with $z \in \Gamma^*$ on the top of the stack by the recursion $\nu(q, \lambda, z) = \lambda$,

$$\nu(q, wb, z) = \nu(q, w, z)\, \nu(\delta_Q(q, w, z), b, \delta_{\Gamma^*}(q, w, z)).$$

The *output* of the compressor C on input $w \in \Sigma^*$ is the string $C(w) = \nu(q_0, w, z_0)$.

The input of an information-lossless compressor can be reconstructed from the output and the final state reached on that input.

Definition 3.4. *A BPDC $C = (Q, \Sigma, \Gamma, \delta, \nu, q_0, z_0, c)$ is information-lossless (IL) if the function*

$$\begin{aligned} \Sigma^* &\to \Sigma^* \times Q \\ w &\mapsto (C(w), \delta_Q(w)) \end{aligned}$$

is one-to-one. An information-lossless pushdown compressor (ILPDC) is a BPDC that is IL.

Intuitively, a BPDC *compresses* a string w if $|C(w)|$ is significantly less than $|w|$. Of course, if C is IL, then not all strings can be compressed. Our interest here is in the degree (if any) to which the prefixes of a given sequence $S \in \Sigma^\infty$ can be compressed by an ILPDC.

Notation. The best-case *pushdown compression ratio* of a sequence $S \in \Sigma^\infty$ is $\rho_{PD}(S) = \inf\{\rho_C(S) \mid \ C \text{ is an ILPDC}\}$.

The worst-case *pushdown compression ratio* of a sequence $S \in \Sigma^\infty$ is $R_{PD}(S) = \inf\{R_C(S) \mid \ C \text{ is an ILPDC}\}$.

As mentioned earlier, the standard PD compression model does not guarantee the decompression to be feasible and it is currently not known whether the exponential time brute force inversion can even be improved to polynomial time. To guarantee both decompression and compression to be feasible, we require the existence of a PD machine that given the compressed string (and the final state), outputs the decompressed one. This yields two PD compression schemes, the standard one (PD) and invertible PD. Contrary to Finite State computation, it is not known whether both are equivalent. This is by no means a limitation, since all results in this paper are always stated in the strongest form, i.e. we obtain results of the form "X beats PD" and "invertible PD beats X".

Here is the definition of invertible PD compressors.

Definition 3.5. *(C, D) is an invertible PD compressor (denoted* invPD*) if C is an ILPDC and D is a PD transducer s.t. $D(C(w), \delta_Q(w)) = w$, i.e. D, given both $C(w)$ and the final state, outputs w.*

In section 4.4 in order to make our result stronger we will also consider PDC that have endmarkers, a characteristic that can achieve a better compression

rate. In this case the information lossless (IL) condition is relaxed to the function

$$\begin{aligned} \Sigma^* &\to \Sigma^* \times Q \\ w &\mapsto (C(w\$), \delta_Q(w)) \end{aligned}$$

being one-one, for \$ a fixed symbol out of Σ.

3.3 Lempel Ziv compression scheme

Let us give a brief description of the classical LZ78 algorithm [16]. Given an input $x \in \Sigma^*$, LZ parses x in different phrases x_i, i.e., $x = x_1 x_2 \dots x_n$ ($x_i \in \Sigma^*$) such that every prefix $y \sqsubset x_i$, appears before x_i in the parsing (i.e. there exists $j < i$ s.t. $x_j = y$). Therefore for every i, $x_i = x_{l(i)} b_i$ for $l(i) < i$ and $b_i \in \Sigma$. We sometimes denote the number of phrases in the parsing of x as $P(x)$. After step i of the algorithm, the i first phrases $x_1, \dots, x_i$ have been parsed and stored in the so-called *dictionary*. Thus, each step adds one word to the dictionary.

LZ encodes x_i by a prefix free encoding of $x_{l(i)}$ and the symbol b_i, that is, if $x = x_1 x_2 \dots x_n$ as before, the output of LZ on input x is

$$LZ(x) = c_{l(1)} b_1 c_{l(2)} b_2 \dots c_{l(n)} b_n$$

where c_i is a prefix-free coding of i (and $x_0 = \lambda$).

LZ is usually restricted to the binary alphabet, but the description above is valid for any alphabet Σ.

4 The performances of the LZ78 algorithm, plogon compressors and pushdown compressors are incomparable

In this section we prove that the new family of compressors we have introduced, plogon, is incomparable with two previously studied restricted memory compression algorithms, that are pushdown compressors and Lempel Ziv compression scheme. That is, on different individual sequences plogon can beat LZ78 and pushdown whereas on others it can be outperformed by those two methods. In all cases we get low worst-case rate (ρ) for one method versus high best-case rate (R) for the other, i.e. the widest possible separation between them. Also regarding the two PD compression schemes (standard PD and invertible PD) we always prove the strongest result possible, i.e. we show results of the form "X beats PD" and "invertible PD beats X".

The comparison of pushdown and Lempel Ziv has been presented in [2].

4.1 plogon beats Lempel Ziv

Our first result uses a Copeland-Erdös sequence [6, 7] on which Lempel-Ziv has maximal compression ratio, whereas with logspace each prefix of the sequence can be completely reconstructed from its length.

Theorem 4.1. *There exists a sequence S such that*

$$R_{\text{plogon}}(S) = 0 \quad \textit{and } \rho_{LZ}(S) = 1.$$

Proof. Let $S = E(\{0,1\}^*) = 0100011011000\ldots$ be the enumeration of binary strings in the standard order. LZ does not compress S at all, for this algorithm it is the worst possible case, i.e.

$$\rho_{LZ}(S) = 1.$$

For any input w, with $|w| = n$, let $m \in \mathbb{N}$, $x \in \{0,1\}^*$ be such that $w = S[1\ldots m]x$, and $S[1\ldots m+1] \not\sqsubset w$. Then we define compressor C as $C(w,|w|) = \text{dbin}(m)01x$, where $\text{dbin}(m)$ is m written in binary with every bit doubled (such that the separator 01 can be recognized). C is clearly 1-1. C is plogon, because on input (w,n), C reads the input online to check that w is a prefix of S (i.e. the standard enumeration of binary strings); the biggest string to check has size $\log n$, therefore the check can be done in plogon. As soon as the check fails, C outputs the length (in binary, with every bit doubled) of the prefix of the input that satisfied the check (at most $2\log n$ bits) followed by 01 and the rest of the input.

The worst case compression ratio for sequence S is given by

$$R_{\text{plogon}}(S) = \limsup_{n\to\infty} \frac{|C(S[1\ldots n],n)|}{n} = \limsup_{n\to\infty} \frac{2\log n}{n} = 0.$$

4.2 Lempel Ziv beats plogon

Our second comparison detects sequences on which Lempel-Ziv achieves optimal compression whereas a plogon compressor has the worst possible performance. The construction is based on repetition of Kolmogorov random strings.

Theorem 4.2. *There exists a sequence S such that*

$$R_{LZ}(S) = 0 \quad \textit{and } \rho_{\text{plogon}}(S) = 1.$$

Proof. Let $A, c \in \mathbb{N}$ with $c \geq 7$. For each $i \in \mathbb{N}$, let R_i be a Kolmogorov random string (i.e. $K(R_i) > i - A$ for A the constant just fixed) with $|R_i| = i$. Let

$$S_n = R_1 R_2^{2^c} R_3^{3^c} \ldots R_n^{n^c}$$

($R_n^{n^c}$ means n^c copies of R_n) and let S be the infinite sequence having all S_n as prefixes.

Lemma 4.3.

$$\frac{|LZ(S_n)|}{|S_n|} \leq \frac{n^{\frac{c+6}{2}}}{n^{c+1}}$$

for n large enough.

Lemma 4.4. *Let* $S_{n,t} = R_1 R_2^{2^c} R_3^{3^c} \dots R_n^{n^c} R_{n+1}^t$ *where* $1 \le t < (n+1)^c$. *Then*

$$\frac{|LZ(S_{n,t})|}{|S_{n,t}|} \le \frac{n^{(c+7)/2}}{n^{c+1}}$$

for n *large enough.*

Lemma 4.5. *For almost every* k, $\frac{|LZ(S[1\dots k]|)}{k} \le k^{(-1+6/c)/2}$ *i.e., for any* $c \ge 7$ $R_{LZ}(S) = 0$.

Let us show that the sequence S is not compressible by ILpCs. For this we show that each large substring x of the input that is a Kolmogorov random word cannot be compressed by a pushdown transducer, independently of the computation performed before processing x.

Let C be an ILpC. For strings z, α, β, x with $z = \alpha x \beta$ and $|z| = m$, denote by $C(s, x, m)$ the output of C starting in configuration s and reading x out of an input of length m. A valid configuration, is a configuration s such that there exists a string c such that $C(s_0, c, m)$ ends in configuration s, where s_0 is the start configuration of C. For example if s is the configuration of C after reading a, then $C(s, x, m)$ is the output of C while reading part x of input $z = axb$. Note that $|s| \le \log(m)^{O(1)}$.

Lemma 4.6. *Let* C *be an ILpC, running in space* $\log^a m$, *and let* $0 < T \le 1$. *Then for every* $d \in \mathbb{N}$ *and almost every* $r \in \mathbb{N}$, *for every random string* $x \in \{0,1\}^r$ *(with* $K(x) \ge T|x| - A$ *for some fixed constant* A*), for every* t *(*$|x| \le t \le |x|^d$*) and for every valid configuration* s *(*$|s| \le \log^a t$*)*

$$|C(s, x, t)| \ge T|x| - \log^{2a} |x|.$$

Proof. Suppose by contradiction that $C(s, x, t) = p$, with $|p| < Tr - \log^{2a} r$; denote by s^x the configuration of C after having read x. Then $p' = (s^x, s, t, r, p)$ (p' is encoded by doubling all bits in s^x, s, t, r, separated by the delimiter 01 followed by p) yields a program for x:
"Find y with $|y| = r$ such that $C(s, y, t) = p$, and C ends in configuration s^x after reading y."

y is unique because otherwise suppose there are two strings y, y' ($|y| = |y'|$) such that $C(s, y, t) = C(s, y', t)$, and C ends in the same configuration on y and y'. Let b be a string that brings C into configuration s. Then for $z = 1^{t-|by|}$ we have $C(byz, t) = C(by'z, t)$ which contradicts C being 1-1. Therefore y is unique, i.e. $y = x$. Thus for r sufficiently large

$$|p'| \le 2(|s^x| + |s| + |t| + |r|) + |p| \le 2(\log^a r^d + \log^a r^d + \log r^d + \log r) + Tr - \log^{2a} r$$

$$\le Tr - \frac{\log^{2a} r}{2}$$

which contradicts the randomness of x.

Lemma 4.7. *Let* C *be an ILpC, running in space* $\log^a m$. *Then for every* $\epsilon > 0$ *and for almost every* m, $\frac{C(S[1\cdots m])}{m} > 1 - \epsilon$ *i.e.,* $\rho_{\text{plogon}}(S) = 1$.

Proof. Let $\epsilon > 0$ and let $\epsilon' = \frac{\epsilon}{4 \cdot 3^{c+2}}$. Let n, t, l ($0 \leq l \leq n$, $0 \leq t < n^c$) be such that $S[1 \dots m] = S_{n-1} R_n^t R_n[1 \dots l]$.

The idea is to apply Lemma 4.6 to $R_{\epsilon' n}^{(\epsilon' n)^c} \dots R_{n-1}^{(n-1)^c} R_n^t R_n[1 \dots l]$. Let d be such that $(\epsilon' n)^d \geq n^{c+2}$ (for all $n \geq 2$), i.e. $(\epsilon' n)^d \geq m$. By Lemma 4.6, C on input $S[1 \dots m]$, will output at least $j - \log^{2a} j$ bits on each R_j ($\epsilon' n \leq j < n$). Therefore

$$|C(S[1 \dots m])| \geq \sum_{j=\epsilon' n}^{n-1} (j - \log^{2a} j) j^c$$

whence

$$\begin{aligned} \frac{|C(S[1 \dots m])|}{m} &\geq \frac{\sum_{j=\epsilon' n}^{n-1} j^c (j - \log^{2a} j)}{\sum_{j=1}^{n-1} j^{c+1} + (t+1)n} \qquad \geq \frac{\sum_{j=\epsilon' n}^{n-1} j^c (j - \alpha j)}{\sum_{j=1}^{n-1} j^{c+1} + (t+1)n} \\ &\geq \frac{(1-\alpha) \sum_{j=\epsilon' n}^{n-1} j^{c+1}}{(1+\alpha') \sum_{j=1}^{n-1} j^{c+1}} \end{aligned}$$

where $\alpha, \alpha' > 0$ can be chosen arbitrarily small (for n large enough). Let $\alpha, \alpha' > 0$ be such that $\frac{1-\alpha}{1+\alpha'} > 1 - \epsilon/2$. Thus

$$\begin{aligned} \frac{|C(S[1 \dots m])|}{m} &\geq \frac{1-\alpha}{1+\alpha'} - \frac{1-\alpha}{1+\alpha'} \cdot \frac{\sum_{j=1}^{\epsilon' n - 1} j^{c+1}}{\sum_{j=1}^{n-1} j^{c+1}} \geq \frac{1-\alpha}{1+\alpha'} - \frac{\epsilon' n^{c+2}}{n/3(n/3)^{c+1}} \\ &= \frac{1-\alpha}{1-\alpha'} - \epsilon' 3^{c+2} > 1 - \epsilon/2 - \epsilon/4 \\ &> 1 - \epsilon \, . \end{aligned}$$

Since ϵ is arbitrary, $\rho_{\text{plogon}}(S) = 1$.

This finishes the proof of Theorem 4.2.

4.3 invPD beats plogon

The following result shows that both invertible pushdown automata and visibly pushdown automata outperform plogon compressors on certain sequences.

Theorem 4.8. *For each $\epsilon > 0$ there exists a sequence S such that*

$$R_{\text{invPD}}(S) \leq 1/2 \quad \textit{and } \rho_{\text{plogon}}(S) \geq 1 - \epsilon.$$

Visibly pushdown automata are defined in [4, 13] and are extensively used in the compression of XML. For these automata the input alphabet has three types of symbols, call symbols, return symbols, and internal symbols. The main restriction is that while reading a call, the automaton must push one symbol, while reading a return symbol, it must pop one symbol (if the stack is non-empty), and while reading an internal symbol, it can only update its control state.

Theorem 4.9. *For each $\epsilon > 0$ there exists a sequence S such that*

$$R_{\text{visiblyPD}}(S) \leq 1/2 \quad \textit{and } \rho_{\text{plogon}}(S) \geq 1 - \epsilon.$$

4.4 plogon beats PD

The next result shows that plogon compressors outperform pushdown compressors on certain sequences.

Theorem 4.10. *There exists a sequence S such that*

$$R_{\text{plogon}}(S) = 0 \quad \textit{and } \rho_{\text{PD}}(S) = 1.$$

This last theorem also holds even when considering that the pushdown compressors input has an endmarker, a characteristic that can achieve a better compression rate.

Theorem 4.11. *There exists a sequence S such that*

$$R_{\text{plogon}}(S) = 0 \quad \textit{and } \rho_{\text{PD with endmarker}}(S) = 1.$$

5 Conclusion

The equivalence of compression ratio, effective dimension, and log-loss unpredictability has been explored in different settings [8, 11, 18]. It is known that for the cases of finite-state, polynomial-space, recursive, and constructive resource-bounds, natural definitions of compression and dimension coincide, both in the case of infinitely often compression, related to effective versions of Hausdorff dimension, and that of almost everywhere compression, matched with packing dimension. The general matter of transformation of compressors in predictors and vice versa is widely studied [19].

In this paper we have done a complete comparison of plogon compression with both pushdown and LZ-compression. It is straightforward to construct a prediction algorithm based on Lempel-Ziv compressor that uses similar computing resources, and it has been proved in [1] that bounded-pushdown compression and dimension coincide. This leaves us with the natural open question of whether each plogon compressor can be transformed into a plogon prediction algorithm, for which the log-loss unpredictability coincides with the compression ratio of the initial compressor, that is, whether the natural concept of plogon dimension coincides with plogon compressibility. A positive answer would get plogon computation closer to pushdown devices, and a negative one would make it closer to polynomial-time algorithms, for which the answer is likely to be negative [17].

References

1. P. Albert, E. Mayordomo, and P. Moser. Bounded pushdown dimension vs lempel ziv information density. Technical Report TR07-051, ECCC: Electronic Coloquium on Computational Complexity, 2007.
2. P. Albert, E. Mayordomo, P. Moser, and S. Perifel. Pushdown compression. In *Proceedings of the 25th Symposium on Theoretical Aspects of Computer Science (STACS 2008)*, pages 39–48, 2008.

3. N. Alon, Y. Matias, and M. Szegedy. The space complexity of approximating the frequency moments. *Journal of Computer and System Sciences*, 58:137–147, 1999.
4. R. Alur and P. Madhusudan. Adding nesting structure to words. In *Proceedings of the Tenth International Conference on Developments in Language Theory*, volume 4036 of *Lecture Notes in Computer Science*. Springer, 2006.
5. J. Autebert, J. Berstel, and L. Boasson. Context-free languages and pushdown automata. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages, volume 1, Word, Language, Grammar*, pages 111–174. Springer-Verlag, 1997.
6. D. G. Champernowne. Construction of decimals normal in the scale of ten. *J. London Math. Soc.*, 2(8):254–260, 1933.
7. A. Copeland and P. Erdös. Note on normal numbers. *Bulletin of the American Mathematical Society*, 52:857–860, 1946.
8. J. J. Dai, J. I. Lathrop, J. H. Lutz, and E. Mayordomo. Finite-state dimension. *Theoretical Computer Science*, 310:1–33, 2004.
9. S. Hariharan and P. Shankar. Evaluating the role of context in syntax directed compression of xml documents. In *Proceedings of the 2006 IEEE Data Compression Conference (DCC 2006)*, page 453, 2006.
10. J. Hartmanis, N. Immerman, and S. Mahaney. One-way log-tape reductions. In *Proceedings of the 19th Annual Symposium on Foundations of Computer Science (FOCS'78)*, pages 65–72. IEEE Computer Society, 1978.
11. J. M. Hitchcock. *Effective Fractal Dimension: Foundations and Applications*. PhD thesis, Iowa State University, 2003.
12. P. Indyk and D. Woodruff. Optimal approximations of the frequency moments of data streams. In *Proceedings of the 37th Annual ACM Symposium on Theory of Computing (STOC 2005)*, pages 202–208. ACM, 2005.
13. V. Kuma, P. Madhusudan, and M. Viswanathan. Visibly pushdown automata for streaming xml. In *International World Wide Web Conference WWW 2007*, pages 1053–1062, 2007.
14. J. I. Lathrop and M. J. Strauss. A universal upper bound on the performance of the Lempel-Ziv algorithm on maliciously-constructed data. In B. Carpentieri, editor, *Compression and Complexity of Sequences '97*, pages 123–135. IEEE Computer Society Press, 1998.
15. C. League and K. Eng. Type-based compression of xml data. In *Proceedings of the 2007 IEEE Data Compression Conference (DCC 2007)*, pages 272–282, 2007.
16. A. Lempel and J. Ziv. Compression of individual sequences via variable rate coding. *IEEE Transaction on Information Theory*, 24:530–536, 1978.
17. M. López-Valdés and E. Mayordomo. Dimension is compression. In *Proceedings of the 30th International Symposium on Mathematical Foundations of Computer Science*, volume 3618 of *Lecture Notes in Computer Science*, pages 676–685. Springer-Verlag, 2005.
18. E. Mayordomo. Effective fractal dimension in algorithmic information theory. In *New Computational Paradigms: Changing Conceptions of What is Computable*, pages 259–285. Springer-Verlag, 2008.
19. D. Sculley and C. E. Brodley. Compression and machine learning: A new perspective on feature space vectors. In *Proceedings of the Data Compression Conference (DCC-2006)*, pages 332–341, 2006.