Theoretical Computer Science 438 (2012) 62–73

Contents lists available at SciVerse ScienceDirect

Theoretical Computer Science

journal homepage: www.elsevier.com/locate/tcs

A linearly computable measure of string complexity

Verónica Becher *, Pablo Ariel Heiber

Departamento de Computación, Facultad de Ciencias Exactas y Naturales, Universidad de Buenos Aires & CONICET, Argentina

ARTICLE INFO

Article history:
Received 10 February 2011
Received in revised form 20 December 2011
Accepted 7 March 2012
Communicated by B. Durand

ABSTRACT

We present a measure of string complexity, called I-complexity, computable in linear time and space. It counts the number of different substrings in a given string. The least complex strings are the runs of a single symbol, the most complex are the de Bruijn strings. Although the I-complexity of a string is not the length of any minimal description of the string, it satisfies many basic properties of classical description complexity. In particular, the number of strings with I-complexity up to a given value is bounded, and most strings of each length have high I-complexity.

1. Introduction

We present a measure of complexity of strings that counts the number of different substrings in a given string. We call it I-complexity. The least complex strings are, as one may expect, the runs of a single symbol. The most complex strings are those having the largest number of different substrings, the de Bruijn strings [5]. A combinatorial definition yields big families of strings with close to maximal complexity (Theorem 34).

The I-complexity is invariant under alphabet permutation, monotone in the prefix ordering of strings, smooth for prefixing or suffixing one symbol, and subadditive for concatenation. The I-complexity of a string is upper bounded by the string length, and most strings have I-complexity close to the maximum (Theorem 38). We also prove that the number of strings with I-complexity up to a given value is bounded (Corollary 31). These properties merit our definition to be considered a measure of string complexity. In fact, except for monotonicity, these are all basic properties of classical *description complexity* (program-size complexity or Kolmogorov complexity) [12,8]. On the other hand, the property of monotonicity is central in the *monotone complexities*, the variants of Kolmogorov complexity independently introduced in [22] and [16] (see [19] for a comparison).

The I-complexity is defined from combinatorial conditions on strings, while description complexity, in each of its varieties, has been defined as a minimal description length for a given description method. So, it is not surprising that the I-complexity is not a measure of information content in the sense of Shannon [18,8]. The same happens to other complexity measures that are not defined as the length of an injective encoding, as the *Lempel–Ziv complexity* [13] (not to be confused with the Lempel–Ziv compression algorithm [21]). Indeed, the I-complexity and the Lempel–Ziv complexity share many properties: their values are close, they have a similar upper bound, and they are both monotone in the prefix ordering and subadditive for concatenation. Also both functions are linearly computable. However, the Lempel–Ziv complexity has two main drawbacks that the I-complexity overcomes. First, there are infinitely many strings with Lempel–Ziv complexity up to any given value. Second, Lempel–Ziv complexity is defined by a purely procedural string description in such a way that it is hard to prove properties. For instance, Jack Lutz's one-bit catastrophe question "Is the Lempel–Ziv complexity of strings s and $0s$ necessarily close?" [14], is still unknown.

It has been hard, if at all possible, to relate description complexities with combinatorial properties of strings. With the I-complexity we aim to narrow the gap between the two views, and to marry the notion of string complexity with string

* Corresponding author. Tel.: +54 11 4576 3359; fax: +54 11 4576 3359.
E-mail addresses: vbecher@dc.uba.ar (V. Becher), pheiber@dc.uba.ar (P.A. Heiber).

0304-3975/$ – see front matter © 2012 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2012.03.007

processing algorithms and data structures. The value of the I-complexity of a given string is obtained by giving a score to each position in a string, according to how many new substrings are given rise by the symbol at that position. The accumulation of the single scores yields a global indicator of the repeated substrings in the given string. Our scoring rule for each position is exactly that used by Ehrenfeucht and Mycielski in [10]. The definition of the I-complexity function is based on a *repetitions array* that allows for algebraic manipulation. This is an array of non-negative integers that indicates, at each position, the length of the largest repeated substring. The repetitions array is indeed a permutation of the well known *Longest Common Prefix* array (*LCP*), the companion data structure of the *suffix array* [15] (Theorem 18). Since the *LCP* array is computable in linear time and space [1] and we can linearly compute the I-complexity from the *LCP* array, the I-complexity is also linearly computable (Theorem 39).

We see the I-complexity as a trade-off of desirable but, thus far, incompatible features. The soundest computable string complexity measure is the *resource bounded Kolmogorov complexity* [6], because it preserves the neat properties of classical Kolmogorov complexity; but it lacks an efficient algorithm to compute it. *Finite state complexities* [7,17], the variants that replaces Turing machines with finite transducers, have the same difficulty. An efficiently computable alternative is the approximation to description complexity by *compression length* using standard compressors, as done by Cilibrasi and Vitanyi in [9], but it is not subadditive for concatenation. For instance, in dictionary compressors such as the Lempel–Ziv compression algorithm [21] the total dictionary size affects the length of the encoding. A related problem occurs in block compressors based on the Burrows Wheeler Transform plus the Move To Front algorithm [1]. While compression length has been successfully used in experimentation, the property of subadditivity is a concern when dealing with large data, or when it is used to approximate information distance [20]. And, more fundamentally, compression length is *only* procedurally defined, so it is difficult to abstract combinatorial properties.

We carried out some experimentation to test the I-complexity on different sorts of data. Unexpectedly, in these inputs the behavior of the I-complexity turned out to be similar to compression length. These experiments are reported in graphical and numerical form in http://kapow.dc.uba.ar/complexity.

The present paper is devoted to introduce the definition of the I-complexity and prove the basic results. Section 2 introduces the repetitions array and develops its properties. Section 3 treats the I-complexity function. Section 4 compares the Lempel–Ziv complexity with the I-complexity. And the last Section 5 proposes a research line on the I-complexity. A calculator of the I-complexity is available in http://kapow.dc.uba.ar/calci.

1.1. Notation

The cardinality of a set $\mathcal{S}$ is denoted as $|\mathcal{S}|$. We denote by $\mathcal{S}^\ell$ the set of sequences of exactly ℓ elements in the set $\mathcal{S}$ and by $\mathcal{S}^* = \bigcup_{\ell\in\mathbb{N}} \mathcal{S}^\ell$ the set of finite sequences of elements in $\mathcal{S}$. Given a sequence $x \in \mathcal{S}^*$, $|x|$ is its length, and its elements are $x[1], \ldots, x[|x|]$. If $1 \le i \le j \le |x|$, $x[i..j]$ is the sequence of length $j - i + 1$ that has $x[i], \ldots, x[j]$ as elements. If $i > j$, $x[i..j]$ is the empty sequence. We write xy for the concatenation of the sequences x and y, and for a rational k, x^k denotes the sequence of length $k|x|$ such that $x^k[i] = x[((i-1) \bmod |x|) + 1]$. We say x is a subsequence of y, and write $x \subseteq y$, iff there are sequences w, z such that $wxz = y$. We denote by x^r the permutation of x with the elements of x in reversed order, $x^r = x[|x|], \ldots, x[1]$. We write elements$(x) \subseteq \mathcal{S}$ to denote the set of elements that appear in a sequence x. For a fixed *finite* alphabet $\mathcal{A}$ of $\alpha = |\mathcal{A}| \ge 2$ symbols, we call the elements of $\mathcal{A}^*$ *strings*. We call the elements of $\mathbb{N}^*$ *arrays*. Along the text, we use lowercase letters close to the beginning of the alphabet (a, b, c, d) to denote elements of $\mathcal{A}$, letters close to the end of the alphabet $(s, t, \ldots)$ to denote strings, letters in the middle (i, j, k, ℓ, n, m) to denote integers and uppercase letters to denote *arrays*. We use x and y occasionally for real numbers. We write $[\![i, j]\!]$ to denote the array A of length $j - i + 1$ such that $A[k] = i + k - 1$ for $1 \le k \le j - i + 1$.

2. The repetitions array

Definition 1 (*Sorting Array*)**.** Given an array A, its *sorting* $\overrightarrow{A}$ is the array of the same length that contains the same elements as A, but in non-decreasing order.

Clearly, $\overrightarrow{A} = \overrightarrow{B}$ iff A is a permutation of B. We introduce a partial order on arrays that we call *domination*, and it is defined pointwise.

Definition 2 (*Domination Order*)**.** An array A is *dominated* by an array B, written $A \preceq B$, iff $|A| = |B|$ and for every $i \in [1, |A|]$, $A[i] \le B[i]$. We write $A \prec B$ if $A \preceq B$ and $A \ne B$.

Observation 3. *For every array A, B:*

1. *If $A \prec B$ then $\overrightarrow{A} \prec \overrightarrow{B}$; and if $A = B$ then $\overrightarrow{A} = \overrightarrow{B}$.*
2. *$A \preceq B$ iff for every i, j, $A[i..j] \preceq B[i..j]$.*

Proof. Direct from Definitions 1 and 2. □

Definition 4 (*Repetitions Arrays*)**.** The forward repetitions array of a given string s, F_s, is an array of length $|s|$ such that $F_s[i] = \max\{\ell : s[i..i+\ell-1] \subseteq s[i+1..|s|]\}$. Analogously, the backward repetitions array of s, B_s, is such that $|B_s| = |s|$ and $B_s[i] = \max\{\ell : s[i-\ell+1..i] \subseteq s[1..i-1]\}$.

Observation 5. *For all strings s, t and i in range:*

1. $F_s[|s|] = B_s[1] = 0$.
2. $F_s[i] = B_{s^r}[|s| - i - 1]$.
3. $F_s[i] \le F_{st}[i]$, $B_t[i] \le B_{st}[|s| + i]$, $F_t[i] = F_{st}[|s| + i]$ *and* $B_s[i] = B_{st}[i]$.
4. $F_s[i] \le F_s[i + 1] + 1$ *and* $B_s[i + 1] \le B_s[i] + 1$.

Proof. Direct from Definition 4. □

The next two lemmas prove invariant properties of the repetitions array. The first, Lemma 6, shows that B_s and F_s are invariant through permutations in the alphabet. Lemma 7 relates a structural property of strings and the values in the repetitions array. It proves that for a given length ℓ, the difference between the number of positions in B_s with values less than ℓ, and the number of different substrings of length ℓ in s, is exactly the length ℓ minus one. This lemma is crucial for the results to come.

Lemma 6. *Given a string s and a function p : elements$(s) \to \mathcal{A}$, let t be such that $|t| = |s|$ and $t[i] = p(s[i])$. Then, $B_s \preceq B_t$ and $F_s \preceq F_t$, and the equality holds only when p is injective.*

Proof. If p is injective, it is clear by definition that $B_s = B_t$. It is also clear that $B_t[i]$ cannot be less than $B_s[i]$ because if $s[i - B_s[i] + 1..i] \subseteq s[1..i - 1]$ then $t[i - B_s[i] + 1..i] \subseteq t[1..i - 1]$ at the same position. Finally, if p is not injective, let $c, d \in$ elements(s) be such that $c \ne d$ and $p(c) = p(d)$. Let i_c be the leftmost occurrence of c in s, and i_d analogously for d. Without loss of generality assume $i_c < i_d$. By definition $B_s[i_c] = B_s[i_d] = 0$ but $B_t[i_d] \ge 1$ because $t[i_d..i_d] = p(d)$ occurs in $t[1..i_d - 1]$ (where $p(c) = p(d)$ appears). Therefore, in this case, $B_t \ne B_s$. By Observation 5.2 the same holds for F. □

Lemma 7 (*Invariant Property of B_s*)**.** *Given a non-empty string s and a length $\ell \le |s|$, let n be the number of different substrings of length ℓ in s and let m be the number of elements in B_s strictly less than ℓ, namely, $n = |\{t \in \mathcal{A}^\ell : t \subseteq s\}|$, $m = |\{i : B_s[i] < \ell\}|$. Then the following holds: $n + \ell - 1 = m$.*

Proof. By induction on the length of s. If $|s| = \ell$, then $n = 1$ and all elements of B_s are less than $|s|$, so $m = |s| = \ell = \ell + n - 1$. If $|s| > \ell$, $B_s = B_{s[1..|s|-1]}B_s[|s|]$ (see Observation 5.3). Let us call n' the number of different substrings of length ℓ in $s[1..|s| - 1]$ and m' the number of elements less than ℓ in $B_{s[1..|s|-1]}$. By inductive hypothesis $n' + \ell - 1 = m'$. Now, if the string $s[|s| - \ell + 1..|s|] \subseteq s[1..|s| - 1]$ then $n = n'$ and, by definition, $B_s[|s|] \ge \ell$, so $m = m'$ and the equation holds. If $s[|s| - \ell + 1..|s|] \not\subseteq s[1..|s| - 1]$, then $n = n' + 1$ and $B_s[|s|] < \ell$, therefore, $m = m' + 1$, so the result also holds. □

Corollary 8. *Given a string s and a positive integer k, the number of times that k occurs in B_s is one more than the difference between the number of different substrings of lengths k and $k + 1$ in s.*

Proof. Let n_k be the number of different substrings of length k in s and m_k the number of elements strictly less than k in B_s. By Lemma 7, $n_k + k - 1 = m_k$ and $n_{k+1} + k + 1 - 1 = m_{k+1}$, therefore the number of times k occurs in B_s is $m_{k+1} - m_k = n_{k+1} - n_k + 1$. □

The next theorem allows us to regard the array representation as giving a global view of the repetitions in a string, independent of the apparently arbitrary direction. After this theorem, we use only B_s because it simplifies notation, but analogous results apply to F_s.

Theorem 9. $\overrightarrow{F_s} = \overrightarrow{F_{s^r}} = \overrightarrow{B_s} = \overrightarrow{B_{s^r}}$.

Proof. We prove $\overrightarrow{B_s} = \overrightarrow{B_{s^r}}$, the other equalities follow by Observations 3 and 5.2. Since $t \subseteq s \Leftrightarrow t^r \subseteq s^r$, the set of substrings of s of any given length has the same cardinality as the set of substrings of s^r of that same length. Applying Corollary 8 to both we obtain that, for each value k, the number of occurrences of k in B_s and B_{s^r} coincide. □

The following example illustrates that not only trivial structural operations, such as reversing or relabeling the alphabet, make two strings to be considered equal in the repetitions view.

Example 10. $B_{cddc} = B_{cdcc} = 0011$.

As expected, the domination relation is subadditive for string concatenation.

Lemma 11 (*Subadditivity for Concatenation*)**.** *For all strings s, t, $B_sB_t \preceq B_{st}$, and equality holds if and only if* elements$(s) \cap$ elements$(t) = \emptyset$.

Proof. Immediate by Observation 5.3. □

2.1. The least and greatest sorted arrays in the domination preorder

The domination relation on sorted repetitions arrays of a given length is a partial order with a least and greatest element. We now prove that such greatest array is the sorted array of the string that consists of run of a single symbol, as c^ℓ.

The least sorted repetitions array is associated to those strings having all their repeated substrings as short as possible. These are exactly the de Bruijn strings of order k when considering lengths of the form $\alpha^k + k - 1$. However, for arbitrary lengths, we need to consider extensions of de Bruijn strings of the closest lower order.

Lemma 12 (*[4], Theorem 1*). *In alphabets with at least three symbols, de Bruijn strings of order k can be extended to order $k+1$. In a two-symbol alphabet, de Bruijn strings of order k cannot be extended to order $k+1$, but they can be extended to order $k+2$.*

The next definition introduces the array Z_ℓ, the sorted repetitions array of (extended) de Bruijn strings for alphabets of at least three symbols. In the case of a two-letter alphabet, for every de Bruijn length ℓ, Z_ℓ also coincides with the sorted arrays of de Bruijn strings. However, since de Bruijn strings of order $k-1$ cannot be extended to order k, but to order $k+1$ (Lemma 12), for many lengths ℓ that are not of the form $2^k + k - 1$, Z_ℓ is not the sorted array of any binary string. But it is quite close to the sorted repetitions array of extended de Bruijn strings, because at just a few positions Z_ℓ is exceeded by $\overrightarrow{B_s}$ by only 1. Then Lemma 14 proves that for $\alpha \geq 3$, Z_ℓ is the least element in the domination partial order of sorted repetitions arrays of length ℓ. The result also holds for $\alpha = 2$ for de Bruijn lengths, and it is closely below the minimum for other lengths.

Definition 13 (*Sorted Repetitions Array of Extended de Bruijn Strings*). Let Z_ℓ be the array of length ℓ such that, for $1 \leq i \leq \ell$, $Z_\ell[i] = k \Leftrightarrow \alpha^k + k \leq i < \alpha^{k+1} + k + 1$.

Note that Z_ℓ is a non-decreasing array, hence $Z_\ell = \overrightarrow{Z_\ell}$.

Lemma 14 (*Tight Bounds of $\overrightarrow{B_s}$*). *For every $s \in \mathcal{A}^*$, $Z_{|s|} \preceq \overrightarrow{B_s} \preceq [\![0, |s| - 1]\!]$, and the bounds are tight.*

Proof. By Observation 5, $B_s[1] = 0$ and $B_s[i+1] \leq B_s[i] + 1$. By induction on i, these two properties imply $B_s[i] \leq i - 1$, which in turn implies $\overrightarrow{B_s} \preceq [\![0, |s| - 1]\!]$. This upper bound is tight because for each $c \in \mathcal{A}$, $B_{c^\ell} = [\![0, |s| - 1]\!]$.

Let us see the lower bound. Let i be a position in $\overrightarrow{B_s}$. Let $k = \max\{j : \alpha^j + j \leq i\}$. By Lemma 7 there are exactly $n + k - 1$ elements strictly less than k in B_s, where n is the number of different substrings of length k in s; therefore, $n+k-1 \leq \alpha^k+k-1$. Since $\overrightarrow{B_s}$ is sorted, this means $\overrightarrow{B_s}[i] \geq k$. Also, by definition $Z_{|s|}[i] = k$. This implies $Z_{|s|}[i] \leq \overrightarrow{B_s}[i]$ for any position i. Let us see that for an alphabet $\mathcal{A}$ with more than two symbols this lower bound is tight. For any given string length ℓ, let k be the smallest integer such that $\alpha^k + k - 1 \geq \ell$. Let s_0 be a non-cyclic de Bruijn string of order $k-1$ in alphabet $\mathcal{A}$ and let s_1 be a de Bruijn string of order k that extends s_0, which, by Lemma 12, exists. So, s_0 contains every substring of length $k-1$ exactly once, and s_1 contains every substring of length k exactly once. Consider $s = s_1[1..\ell]$. Note that $s_0 \subseteq s$. Since s does not contain any substring of length k more than once, all values in B_s are strictly less than k. Also, every substring of length $\ell \leq k-1$ appears, so there are a maximum of α^ℓ substrings of length ℓ, and there is also a maximum of $|s| - k + 1$ substrings of length k. Applying Corollary 8 it follows that the number of occurrences of each value in B_s and in $Z_{|s|}$ is the same. □

2.2. The repetitions array is a permutation of the Longest Common Prefix

We show that the repetitions array B_s is a permutation of the *Longest Common Prefix* array, the companion data structure of the *suffix array* [15]. A suffix array represents a lexicographically sorted list of all the suffixes in a string, and the *LCP* array stores the lengths of the longest common prefixes of adjacent suffixes in the suffix array. These two data structures are so well known because they allow many string processing problems to be solved in optimal time and space.

Definition 15 (*Suffix Array S_s [15]*). The suffix array S_s of a string s is an array of length $|s|$ such that for every i, $s[S_s[i]..|s|]$ is lexicographically less than $s[S_s[i+1]..|s|]$.

Definition 16 (*LCP Array*). For a non-empty string s, LCP_s is an array of length $|s| - 1$ such that for each i, $LCP_s[i] = \max\{\ell : s[S_s[i]..S_s[i] + \ell - 1] = s[S_s[i+1]..S_s[i+1] + \ell - 1]\}$.

The next lemma proves that the *LCP* array possesses an analogous invariance property as that we proved for the repetitions array in Lemma 7. Since the length of LCP_s is $|s| - 1$ instead of $|s|$, it misses one position, and this is reflected in the invariance property.

Lemma 17 (*Invariance Property of LCP*). *Given a non-empty string s and a length $\ell \leq |s|$, let n be the number of different substrings of length ℓ in s and, let m be the number of elements in LCP_s strictly less than ℓ, namely, $n = |\{t \in \mathcal{A}^\ell : t \subseteq s\}|$, $m = |\{i : LCP_s[i] < \ell\}|$. Then the following holds: $n + \ell - 2 = m$.*

Proof. Fix ℓ. Consider a relation $\sim$ between suffixes of s such that $s[i..|s|] \sim s[j..|s|] \Leftrightarrow (i, j \leq |s| - \ell + 1 \wedge s[i..i+\ell-1] = s[j..j+\ell-1])$. Clearly, $\sim$ is an equivalence relation, and let k be the number of classes. All elements in each equivalence class are consecutive in S_s. Also, by definition, the length of the longest common prefix of two consecutive suffixes in the lexicographic order is strictly less than ℓ if and only if they belong to different classes. Therefore, $m = k - 1$. Note that there are exactly $\ell - 1$ suffixes of length less than ℓ, and necessarily each of them forms a singleton class. Each of the other classes must be represented by a different substring of length ℓ, which is a prefix of all the suffixes in the class. Therefore, $k = \ell - 1 + n$. Putting all together $m = k - 1 = \ell - 1 + n - 1 = n + \ell - 2$. □

Theorem 18. $[\![0, 0]\!]\overrightarrow{LCP_s} = \overrightarrow{B_s}$.

Proof. Put together Lemma 7 and Lemma 17 and observe that every value except 0 occurs the same number of times in LCP_s as in B_s. There is exactly one more 0 in B_s. □

3. The I-complexity function

The function I is defined so as to have smaller values on strings with more and longer repeated substrings. For a given string, the value of I is the accumulation of a score given to each string position, and this score is inversely related to the length of the repetition raised by the symbol at that position, namely, the value of the repetitions array in that position. To achieve the wanted logarithmic complexity of strings of the form c^ℓ we use the discrete derivative function of log, dlog, for each given base n.

Definition 19 (*Discrete Derivative Function of* log)**.** $\operatorname{dlog}_n(x) = \log_n(x+1) - \log_n(x)$.

Observation 20. *For every positive x, $1/(x+1) < (\ln n)\operatorname{dlog}_n x < 1/x$.*

Proof. $(\ln n)\operatorname{dlog}_n x = \ln(x+1) - \ln x = \int_x^{x+1} 1/y\, dy \;<\; 1/x \int_x^{x+1} 1\, dy = 1/x.$
$(\ln n)\operatorname{dlog}_n x = \ln(x+1) - \ln x = \int_x^{x+1} 1/y\, dy \;>\; 1/(x+1) \int_x^{x+1} 1\, dy = 1/(x+1).$ □

From Observation 20 follows that dlog_n is a non-increasing function.

Definition 21 (*I-complexity Function*)**.** The complexity function $I : \mathbb{N}^* \to \mathbb{R}$ is defined as follows: for any repetitions array A,

$$I(A) = \sum_{i=1}^{|A|} \operatorname{dlog}_\alpha(A[i]+1).$$

The complexity function $I : \mathcal{A}^* \to \mathbb{R}$ is defined from its counterpart on arrays: for any string s,

$$I(s) = I(B_s).$$

Observation 22. *For every array A, B, if $A \prec B$ then $I(A) > I(B)$, and if $A \preceq B$ then $I(A) \geq I(B)$.*

Proof. Immediate from the definitions. □

Observation 23. $I(\llbracket x, x+\ell-1 \rrbracket) = \log_\alpha(x+\ell+1) - \log_\alpha(x-1)$.

Proof. $I(\llbracket x, x+\ell-1 \rrbracket) = \sum_{i=x}^{x+\ell-1} \operatorname{dlog}_\alpha(i+1) = \log_\alpha(x+\ell+1) - \log_\alpha(x-1)$. □

Theorem 24. *If $\ell \geq \alpha$, then $I(Z_\ell) < 2\ell/\ln \ell$.*

Proof. By induction on ℓ. Firstly, $Z_\alpha = \llbracket 0, 0 \rrbracket^\alpha$, therefore, $I(Z_\alpha) = \alpha \operatorname{dlog}_\alpha 1 = \alpha \log_\alpha 2 = \alpha/\log_2 \alpha < \alpha/\ln \alpha$. A simple check shows that $I(Z_\ell) < 2\ell/\ln \ell$, for each $\ell \leq 20$ and each $\alpha \leq \ell$. Thus, in the sequel assume $\ell > \max(\alpha, 20)$. Let k be the largest integer such that $\alpha^k + k \leq \ell$. So, $\ell \leq \alpha^{k+1} + k$. Note that $Z_\ell = Z_{\ell-1}\llbracket k, k \rrbracket$. By the inductive hypothesis,

$$I(Z_\ell) = I(Z_{\ell-1}) + \operatorname{dlog}_\alpha(k+1) < \frac{2(\ell-1)}{\ln(\ell-1)} + \frac{1}{(k+1)\ln\alpha}.$$

We give an upper bound of $\log_\alpha \ell$ to derive an upper bound of $\frac{1}{(k+1)\ln\alpha}$:

$$\log_\alpha \ell = \sum_{i=1}^{k} \operatorname{dlog}_\alpha(\ell - i) + \log_\alpha(\ell - k) < \frac{k}{(\ell-k)\ln\alpha} + \log_\alpha(\alpha^{k+1}) = \frac{k}{(\ell-k)\ln\alpha} + k + 1.$$

Then, $\log_\alpha \ell - \frac{k}{(\ell-k)\ln\alpha} < k+1$; hence, $\frac{1}{(k+1)\ln\alpha} < \frac{\ell-k}{(\ell-k)\ln\ell - k}$.
Since $\ell \leq (\ell-k)\ln\ell$, then $\frac{\ell-k}{(\ell-k)\ln\ell-k} \leq \frac{\ell}{(\ell-k)\ln\ell}$.
Thus, $\frac{1}{(k+1)\ln\alpha} < \frac{\ell}{(\ell-k)\ln\ell}$.
To prove $I(Z_\ell) < 2\ell/\ln\ell$ it suffices to show $\frac{2(\ell-1)}{\ln(\ell-1)} + \frac{\ell}{(\ell-k)\ln\ell} \leq \frac{2\ell}{\ln\ell}$.
This is equivalent to $\frac{2\ell}{\ln(\ell-1)} - \frac{2\ell}{\ln\ell} \leq \frac{2}{\ln(\ell-1)} - \frac{\ell}{(\ell-k)\ln\ell}$, which, in turn, it is equivalent to $2\ell \operatorname{dlog}_e(\ell-1) \leq \frac{2(\ell-k)\ln\ell - \ell\ln(\ell-1)}{\ell-k}$.
Using the upper bound of dlog_e it suffices to show

$$\frac{2\ell}{\ell-1} \leq 2\ln\ell - \frac{\ell\ln(\ell-1)}{\ell-k} \;\Leftrightarrow\; \ell\ln(\ell-1)\left(1 + \frac{k-1}{\ell-k}\right) + 2\ell + 2\ln\ell \leq 2\ell\ln\ell.$$

This last inequality holds because $1 + \frac{k-1}{\ell-k} + \frac{2}{\ln(\ell-1)} + \frac{2}{\ell} < 2$, for our assumption $\ell \geq 20$, and k the maximum integer such that $\alpha^k + k \leq \ell$. □

Theorem 25 (*Basic Properties of I-complexity*)**.** *For all strings s, t and symbol c,*

1. $\log_\alpha(|s|+1) \leq I(s)$.
2. *If $|s| \geq \alpha$ then $I(s) < 2|s|/\ln|s|$.*

3. $\max(I(s), I(t)) \leq I(st) \leq I(s) + I(t)$.
4. $I(s) \leq I(cs) \leq I(s) + 1$ *and* $I(s) \leq I(sc) \leq I(s) + 1$.

Proof. For 1 and 2, apply Definition 21, $I(s) = I(B_s) = I(\overrightarrow{B_s})$ and the bounds for $\overrightarrow{B_s}$ in Lemma 14, together with Observation 22. By Observation 23 and Theorem 24, $\log_\alpha(|s|+1) = I([\![0, |s|-1]\!]) \leq I(B_s) \leq I(Z_{|s|}) < 2\,|s|/\ln|s|$. Items 3 and 4 follow directly from the definitions and Lemma 11. □

Observation 26 (*Families of Maximal and Minimal I-complexity*)**.** *Among all strings of the same length, runs c^ℓ have minimal I-complexity and extended de Bruijn strings have maximal I-complexity.*

Proof. Immediate from Lemma 14. □

3.1. On the number of strings with I-complexity up to a given value

To obtain an upper bound of the number of strings with complexity up to a given value k, we give an upper bound of the number of symbols needed to encode a string with complexity up to k. We first give a negative result, Theorem 27, which proves that no encoding length is majorized by a linear function of the I-complexity. The positive result, Corollary 31, gives an encoding whose length is majorized by a quadratic function of the I-complexity. We use the customary asymptotic $\mathcal{O}(.)$ notation to describe the growth rate of the functions.

Theorem 27. $\log|\{s \in \mathcal{A}^* : I(s) \leq k\}| \notin \mathcal{O}(k)$.

Proof. By way of contradiction suppose there is some $n \geq 1$ such that

$$\log_\alpha|\{s \in \mathcal{A}^* : I(s) \leq k\}| < nk.$$

Let $\ell = \alpha^{2n/\ln\alpha}$. If $|s| \leq \ell$ then, by Theorem 25, $I(s) < 2|s|/\ln|s| \leq \ell/n$. Therefore, $\{s \in \mathcal{A}^* : |s| \leq \ell\} \subseteq \{s \in \mathcal{A}^* : I(s) \leq \ell/n\}$. Now, $\log_\alpha|\{s \in \mathcal{A}^* : |s| \leq \ell\}| > \ell$, but $\log_\alpha|\{s \in \mathcal{A}^* : I(s) \leq \ell/n\}| < n\ell/n = \ell$, which is a contradiction. □

Now let us briefly recall the encoding presented by Lempel and Ziv in [13]. A *production* is a transformation of a string that consists on appending a copy of a number of previously occurring consecutive symbols plus a single extra symbol. In this way, a production can be encoded as a triple (i, ℓ, c), where i indicates the position of the source string to start copying, ℓ indicates how many symbols to copy and c is the symbol to append at the end. When the production (i, ℓ, c) is applied to s the result is $s(s[i..|s|]^{\ell/(|s|-i+1)})c$. A string can be described completely by a *production history*, that is, a list of such triples that, when iteratively applied, generate the string from the empty string. Note that every string has a production history, and usually many.

Definition 28 (*[13]*)**.** Given a string, its *exhaustive production history* is such that every production, except possibly the last one, copy as many symbols as possible.

Lemma 29. *The exhaustive production history of a string s consists of at most $2(\log_2\alpha)I(s)$ productions.*

Proof. Let n be the length of the exhaustive production history of s. If $n = 1$, then $|s| = 1$ and $I(s) = \log_\alpha 2 = 1/\log_2\alpha$, so the inequality trivially holds. Assume now $n > 1$ and thus $|s| > 1$. For $1 \leq i \leq n$, let p_i be the index of the first symbol produced by production i in the exhaustive production history of s. For completion, let $p_{n+1} = |s| + 1$. Let $\ell_i = p_{i+1} - p_i$ be the length of each production, and set $\ell_0 = 0$. By definition of exhaustive productions, in production i the substring $s[p_i..p_{i+1} - 1]$ does not occur in $s[1..p_{i+1} - 2]$, with the possible exception of the last production. This means that for every i such that $1 \leq i < n$, $B_s[p_{i+1} - 1] < \ell_i$. Applying Observation 5.4 we can say that $B_s[p_{i+1}] \leq \ell_i$. By Observation 5.1 $B_s[p_1] = 0 \leq \ell_0$, so for every i such that $1 \leq i \leq n$, $B_s[p_i] \leq \ell_{i-1}$. Then, using Observation 5.4 again we obtain that for each j, $B_s[p_i + j] \leq \ell_{i-1} + j$ and $B_s[p_i..p_{i+1} - 1] \preceq [\![\ell_{i-1}, \ell_{i-1} + \ell_i - 1]\!]$.

$$\begin{aligned}
I(s) &= I(B_s) = \sum_{i=1}^{n} I(B_s[p_i..p_{i+1} - 1]) \\
&\geq \sum_{i=1}^{n} I([\![\ell_{i-1}, \ell_{i-1} + \ell_i - 1]\!]) = \sum_{i=1}^{n} \log_\alpha(\ell_{i-1} + \ell_i + 1) - \log_\alpha(\ell_{i-1} + 1). \\
2I(s) &\geq \sum_{i=1}^{n} 2\log_\alpha(\ell_{i-1} + \ell_i + 1) - 2\log_\alpha(\ell_{i-1} + 1) \\
&= 2\log_\alpha(\ell_{n-1} + \ell_n + 1) - \log_\alpha(\ell_0 + 1) - \log_\alpha(\ell_{n-1} + 1) \\
&\quad + \sum_{i=1}^{n-1} \log_\alpha(\ell_{i-1} + \ell_i + 1) - \log_\alpha(\ell_{i-1} + 1) + \log_\alpha(\ell_{i-1} + \ell_i + 1) - \log_\alpha(\ell_i + 1) \\
&= \log_\alpha\left(\frac{(\ell_{n-1} + \ell_n + 1)^2}{(\ell_{n-1} + 1)}\right) + \sum_{i=1}^{n-1} \log_\alpha\left(\frac{(\ell_{i-1} + \ell_i + 1)^2}{(\ell_{i-1} + 1)(\ell_i + 1)}\right) \\
&\geq \log_\alpha 2 + (n-1)\log_\alpha 2 = n\log_\alpha 2.
\end{aligned}$$

The last inequality holds since $\frac{(x+y+1)^2}{(x+1)(y+1)} \geq 2$ for all integers $x, y \geq 1$. Hence, $n \leq 2\,I(s)(\log_\alpha 2)^{-1} = 2(\log_2\alpha)I(s)$. □

Theorem 30. $\log|\{s \in \mathcal{A}^* : |s| \le \ell \wedge I(s) \le k\}| \in \mathcal{O}(k \log(\ell+1))$.

Proof. Let $s \in \{s \in \mathcal{A}^* : |s| \le \ell \wedge I(s) \le k\}$. By Lemma 29, there is a production history of s that consists of at most $\mathcal{O}(k)$ productions. Each production can be encoded in $\mathcal{O}(\log(\ell+1))$ symbols because it consists of two integers not greater than ℓ and a symbol of $\mathcal{A}$. □

Corollary 31. $\log|\{s \in \mathcal{A}^* : I(s) \le k\}| \in \mathcal{O}(k^2)$.

Proof. By Theorem 25.1 $I(s) \ge \log_\alpha(|s|+1)$, which means that $\{s \in \mathcal{A}^* : I(s) \le k\} = \{s \in \mathcal{A}^* : |s| \le \alpha^k - 1 \wedge I(s) \le k\}$. Application of Theorem 30 setting $\ell = \alpha^k - 1$ yields $\log|\{s \in \mathcal{A}^* : I(s) \le k\}| \in \mathcal{O}(k^2)$. □

3.2. A large family of strings with almost maximal complexity

As shown in Observation 26, the strings the with maximal I-complexity are the (extended) de Bruijn strings. Their cardinality the total number of strings of the given de Bruijn length As proved in [5], the cardinality of the acyclic de Bruijn strings of order k in alphabet $\mathcal{A}$, which have length $\alpha^k + k - 1$, is $\alpha!^{\alpha^{k-1}}/\alpha^k$. Here we define, for each length, a family of strings with almost maximal I-complexity and we prove that its cardinality is much larger than that of the de Bruijn family. We actually prove that for large lengths ℓ, the cardinality of this family is close to the total number of strings of length ℓ, while its elements keep a high I-complexity.

Recall that the strings of a given length with high I-complexity are exactly the strings with a minimal sorted repetitions array. We now relax this condition and define a subset of the strings with almost maximal complexity as those having a close to minimal sorted repetitions array. In a de Bruijn string of order k, each string of length k occurs just once. Hence, all values of the repetitions array of a de Bruijn string must be smaller than k. We weaken this property requiring it only at positions with an index multiple of k.

Definition 32 (*Bounded Repetitions Family*)**.** For given integers ℓ and $k \ge \log_\alpha \ell$, we define the set of strings $\mathcal{B}_\ell^k = \{s \in \mathcal{A}^\ell : \forall i \le \lfloor \ell/k \rfloor \ B_s[ik] < k\}$.

As the following result shows, the strings in the family $\mathcal{B}_\ell^k$ contain no repeated substring of length $2k-1$, and the repetitions array of each string is upper bounded as follows.

Observation 33. *Let $s \in \mathcal{B}_\ell^k$. Then, $B_s \preceq [\![0, k-2]\!]([\![k-1, 2k-2]\!]^{(\ell-k+1)/k})$.*

Proof. First observe that $B_s[1..k-1] \preceq [\![0, k-2]\!]$ is true for any string s. By definition of $\mathcal{B}_\ell^k$, $B_s[ik] \le k-1$. Applying Observation 5.4 it follows that $B_s[ik+j] \le k-1+j$ for every j, which directly implies the result. □

Theorem 34 (*Strings in $\mathcal{B}_\ell^k$ Have Almost Maximal Complexity*)**.** *If $s \in \mathcal{B}_\ell^k$ then*

$$I(s) \ge \frac{\ell - k + 1}{k} \log_\alpha 2 + \log_\alpha k.$$

Proof. Let $s \in \mathcal{B}_\ell^k$. By Definition $I(s) = I(B_s)$, and by Observations 22 and 33,

$$\begin{aligned} I(B_s) &\ge I([\![0, k-2]\!][\![k-1, 2k-2]\!]^{(\ell-k+1)/k}) \\ &\ge \sum_{i=0}^{k-2} \mathrm{dlog}_\alpha(i+1) + \frac{\ell-k+1}{k} \sum_{i=k-1}^{2k-2} \mathrm{dlog}_\alpha(i+1) \\ &= \log_\alpha k + (\ell-k+1)/k\,(\log_\alpha 2k - \log_\alpha k) = \log_\alpha k + (\log_\alpha 2)(\ell-k+1)/k. \end{aligned}$$ □

Theorem 35 ($\mathcal{B}_\ell^k$ *Has a Large Number of Elements*)**.** $|\mathcal{B}_\ell^k| > \alpha^{\ell(1-(k\ln\alpha)^{-1})}$.

Proof. We give a lower bound for $|\mathcal{B}_\ell^k|$ by bounding the number of possibilities to construct a string $s \in \mathcal{B}_\ell^k$. We do this in $\lceil \ell/k \rceil$ steps, choosing k symbols at step $i \le \lfloor \ell/k \rfloor$ and $\ell \bmod k$ symbols in the final step, if it exists. Let s_i be the string of length k chosen at step $i \le \lfloor \ell/k \rfloor$ and w be the (possibly empty) string of symbols of length $\ell \bmod k$ chosen in the final step. The constructed string is then $s = s_1 s_2 \ldots s_{\lfloor \ell/k \rfloor} w$. To ensure $B_s[ik] < k$ all we need is to choose s_i such that $s_i \not\subseteq s_1 s_2 \ldots s_{i-1} s_i[1..k-1]$. For s_1 we can choose any k elements, so there are α^k possibilities. There are at most $(i-2)k+1$ choices that result in $s_i \subseteq s_1 s_2 \ldots s_{i-1}$. The other forbidden choices are such that $(s_{i-1}[2..k]s_i)[j..j+k-1] = s_i$. Note that, for fixed j, s_i is univocally defined. Therefore, the number of forbidden choices in this case is at most the number of choices for j, which are $k-1$. Overall, we have at most $(i-1)k$ forbidden choices at step i, which implies that there are at least $\alpha^k - ik + k$ valid possibilities. Finally, w can be chosen in any way, so there are $\alpha^{n \bmod k}$ possibilities for w. This allows us to construct at least this number of different strings $s \in \mathcal{B}_\ell^k$:

$$\alpha^{\ell \bmod k} \prod_{i=1}^{\lfloor \ell/k \rfloor} \alpha^k - ik + k = \alpha^{\ell \bmod k} \prod_{i=0}^{\lfloor \ell/k \rfloor - 1} \alpha^k - ik.$$

We give a lower bound for $\log_\alpha |\mathcal{B}_\ell^k|$ approximating a sum of logarithmic expressions with their integration. To be sound we should take the limit of the integral in interval $[0, \ell/k]$, because it may include the logarithm of zero in its border. In the limit they are all null because they are multiplied by factors that converge faster to 0. In the next inequalities, we abuse notation to avoid such technicalities and make the proof easier to follow.

$$\begin{aligned}
\log_\alpha |\mathcal{B}_\ell^k| &\geq \log_\alpha \left(\alpha^{\ell \bmod k} \prod_{i=0}^{\lfloor \ell/k \rfloor - 1} \alpha^k - ik \right) = \ell \bmod k + \sum_{i=0}^{\lfloor \ell/k \rfloor - 1} \log_\alpha(\alpha^k - ik) \\
&\geq \int_{\lfloor \ell/k \rfloor}^{\ell/k} k \, dx + \int_0^{\lfloor \ell/k \rfloor} \log_\alpha(\alpha^k - xk) \, dx \\
&\geq \int_0^{\ell/k} \log_\alpha(\alpha^k - xk) \, dx \\
&\geq \frac{(\alpha^k - \ell)\ln(\alpha^k - \ell) - \alpha^k + \ell - \alpha^k \ln \alpha^k + \alpha^k}{-k \ln \alpha} \\
&\geq \frac{(\alpha^k - \ell)\ln(\alpha^k - \ell) + \ell}{-k \ln \alpha} + \alpha^k \\
&\geq -\alpha^k + \ell - \frac{\ell}{k \ln \alpha} + \alpha^k = \ell(1 - (k \ln \alpha)^{-1}).
\end{aligned}$$

Hence, there are at least $\alpha^{\ell(1-(k\ln\alpha)^{-1})}$ strings in $\mathcal{B}_\ell^k$. □

When k increases, the bound for I-complexity of the strings in $\mathcal{B}_\ell^k$ decreases and the total number $|\mathcal{B}_\ell^k|$ slowly increases. The family associated to the highest complexity is obtained using a small value $k = \lceil \log_\alpha \ell \rceil$.

Corollary 36 ($\mathcal{B}_\ell^{\lceil \log_\alpha \ell \rceil}$ *is a Large Family of Almost-Maximal Complexity*)**.** *For* $s \in \mathcal{B}_\ell^{\lceil \log_\alpha \ell \rceil}$,

1. $|s| = \ell$ *and* s *contains no repeated substrings of length* $2\lceil \log_\alpha \ell \rceil - 1$.
2. $I(s) > \log_\alpha 2\,\ell/\lceil \log_\alpha \ell \rceil$, *which is greater than* $\ln 2/2 = 0.34657..$ *times the upper bound of the* I*-complexity of strings of that length.*
3. *The cardinality of* $\mathcal{B}_\ell^{\lceil \log_\alpha \ell \rceil}$ *is close to the maximum* α^ℓ: $|\mathcal{B}_\ell^{\lceil \log_\alpha \ell \rceil}| \geq \alpha^{\ell(1-(\ln \ell)^{-1})}$. *Thus, for any* $\varepsilon > 0$ *and sufficiently large* ℓ, $\mathcal{B}_\ell^{\lceil \log_\alpha \ell \rceil}$ *has more than* $\alpha^{\ell(1-\varepsilon)}$ *elements.*

Proof. Immediate from Theorems 34 and 35. □

3.3. Most strings have high I-complexity

On Theorem 25.2 we proved $2I(s) < 2|s|/\ln|s|$. For sufficiently long strings, we can tighten this result. This is to be compared with the expected value of the I-complexity given in Theorem 38.

Theorem 37 (*Tighter Upper Bound for Large Strings*)**.** $I(s) \leq |s|/\ln|s| + \mathcal{O}(|s|/\ln^2|s|)$.

Proof. Let k be the largest integer such that $\alpha^k + k \leq |s|$. By Definition 13, each i such that $0 \leq i \leq k-1$, occurs $\alpha^{i+1} - \alpha^i + 1$ times in $Z_{|s|}$, k occurs $|s| - \alpha^k - k + 1$ times in $Z_{|s|}$, and other values do not occur at all. Therefore,

$$I(s) \leq I(Z_{|s|}) = (|s| - \alpha^k - k + 1)\,\mathrm{dlog}_\alpha(k+1) + \sum_{i=0}^{k-1} (\alpha^{i+1} - \alpha^i + 1)\,\mathrm{dlog}_\alpha(i+1).$$

Using $(\ln \alpha)\,\mathrm{dlog}_\alpha(x) < 1/x$ and rearranging the sum we obtain:

$$\begin{aligned}
I(s)\ln\alpha &< (|s| - \alpha^k - k + 1)/(k+1) + \log_\alpha(k+1) + \textstyle\sum_{i=1}^k (\alpha^i - \alpha^{i-1})/i \\
&\leq (|s| - k + 1)/(k+1) - 1 + \log_\alpha(k+1) + \textstyle\sum_{i=1}^k \alpha^i/i - \alpha^i/(i+1).
\end{aligned}$$

The summation is easily bounded as follows:

$$\sum_{i=1}^{k} \frac{\alpha^i}{i(i+1)} \leq \sum_{i=1}^{\lceil k/2 \rceil} \frac{\alpha^i}{i(i+1)} + \frac{\alpha^{i+\lceil k/2 \rceil}}{(i+\lceil k/2 \rceil)(i+\lceil k/2 \rceil+1)} \leq \sum_{i=1}^{\lceil k/2 \rceil} 2\frac{\alpha^{i+\lceil k/2 \rceil}}{\lceil k/2 \rceil^2} \leq 8\alpha^{k+2}/k^2.$$

Using this result, $I(s)\ln\alpha \leq |s|/(k+1) + \log_\alpha(k+1) + 8\alpha^2|s|/k^2$. Since we assumed k to be the largest integer such that $\alpha^k + k \leq |s|$, we have $\log_\alpha|s| < k+2$. We conclude,

$$\begin{aligned}
I(s)\ln\alpha &\leq |s|/(\log_\alpha|s| - 1) + \log_\alpha(k+1) + 8\alpha^2|s|/(\log_\alpha|s| - 2)^2 \\
&\leq |s|/\log_\alpha|s|(1 + 1/(\log_\alpha|s| - 1)) + \mathcal{O}(|s|/\ln^2|s|).
\end{aligned}$$

Hence, $I(s) \leq |s|/\ln|s| + \mathcal{O}(|s|/\ln^2|s|)$. □

The next theorem gives a lower bound of the expected value of the I-complexity function on the strings of a given length for the equiprobably Bernoulli distribution.

Theorem 38 (*The Expected Value of I is in the Order of Magnitude of the Maximum*)**.**

$$\alpha^{-\ell} \sum_{s\in\mathcal{A}^\ell} I(s) \geq \ell/\log_2 \ell - \mathcal{O}(\ell/\log_2{}^2 \ell).$$

Proof. Assume $\ell > \alpha$. For any s such that $|s| = \ell$, and any given $k \leq \ell$, we can factorize the repetitions array B_s in $m = \lfloor \ell/k \rfloor$ blocks and a tail of length $1 + \ell \bmod k$ in this way:

$$B_s = B_s[1..k-1]B_s[k..2k-1]...B_s[(m-1)k..mk-1]B_s[mk..\ell].$$

Then, $I(s) = I(B_s[1..k-1]) + \sum_{i=1}^{m-1} I(B_s[ik..(i+1)k-1]) + I(B_s[mk..\ell])$. By the same reasoning as in the proof of Theorem 35, for each $i \leq m-1$, we can choose the symbols in all positions except the range $[ik-k+1, ik]$ and then we still have at least $\alpha^k - ik + k$ choices that lead to $B_s[ik] < k$. This means there are at least $\alpha^{\ell-k}(\alpha^k - ik + k)$ strings in which $B_s[ik] < k$, $B_s[ik..(i+1)k-1] \preceq [\![k-1, 2k-2]\!]$ and $I(B_s[ik..(i+1)k-1]) \geq \log_\alpha 2$. Also, for all s, $B_s[1..k-1] \preceq [\![0, k-2]\!]$ and $I(B_s[1..k-1]) \geq \log_\alpha k$. Therefore,

$$\begin{aligned}
\sum_{s\in\mathcal{A}^\ell} I(s) &= \sum_{s\in\mathcal{A}^\ell} \left(I(B_s[1..k-1]) + \sum_{i=1}^{m-1} I(B_s[ik..(i+1)k-1]) + I(B_s[mk..\ell]) \right) \\
&\geq \alpha^\ell \log_\alpha k + \sum_{i=1}^{m-1} \alpha^{\ell-k}(\alpha^k - ik + k) \log_\alpha 2. \\
\alpha^{-\ell} \sum_{s\in\mathcal{A}^\ell} I(s) &\geq \log_\alpha k + \sum_{i=0}^{m-2} (1 - \alpha^{-k} ik) \log_\alpha 2 \\
&\geq \log_\alpha k + (m-1)(1 - \alpha^{-k} mk) \log_\alpha 2 \\
&\geq \ell/k\,(1 - \alpha^{-k}\ell) \log_\alpha 2 \qquad \text{(assuming } k \geq 4\text{)}.
\end{aligned}$$

Now, let $k = \lfloor \log_\alpha \ell + \log_\alpha \log_\alpha \ell \rfloor$,

$$\begin{aligned}
\alpha^{-\ell} \sum_{s\in\mathcal{A}^\ell} I(s) &\geq \ell/(\log_\alpha \ell + \log_\alpha \log_\alpha \ell)(1 - \alpha^{-(\log_\alpha \ell + \log_\alpha \log_\alpha \ell - 1)}\ell) \log_\alpha 2 \\
&\geq \ell/(\log_\alpha \ell + \log_\alpha \log_\alpha \ell)(1 - \alpha/\log_\alpha \ell) \log_\alpha 2 \\
&\geq \ell/\log_2 \ell (1 - \log_\alpha \log_\alpha \ell/(\log_\alpha \ell + \log_\alpha \log_\alpha \ell))(1 - \alpha/\log_\alpha \ell). \quad \square
\end{aligned}$$

We conclude that, for each given length, necessarily most strings have I-complexity close to the maximum. Figs. 1 and 2 depict how the distribution of the I-complexity looks in practice.

3.4. I is computable in linear time and space

The following theorem shows that the function I can be calculated efficiently. The calculation method is straightforward given a method to compute the *LCP* array, for which there is extensive research and available implementations (see [11] and references in [1]).

Theorem 39. *I is computable in linear time and space.*

Proof. By Theorem 18, $I(s) = I(B_s) = I([\![0, 0]\!]LCP_s)$. Since LCP_s is computable linear time and space, the function I can be computed linearly as well. $\square$

4. Comparison with Lempel–Ziv complexity

In [13] Lempel and Ziv define the complexity $L : \mathcal{A}^* \to \mathbb{N}$, as a function that counts the number of operations needed to reconstruct a string from its production history, a concept that we already introduced in Section 3. $L(s)$ is the length of a shortest production history of s. In the same work, they prove the following result.

Theorem 40 (*[13], Theorem 1*)**.** *$L(s)$ is equal to the length of the exhaustive production history of s.*

4.1. Main differences between Lempel–Ziv complexity and I-complexity

As witnessed by the next example, with the exception of very short lengths, Lempel–Ziv complexity does not have a proper lower-bound of the complexity of the strings of a given length. It also shows that this complexity is not invariant by reversing the string.

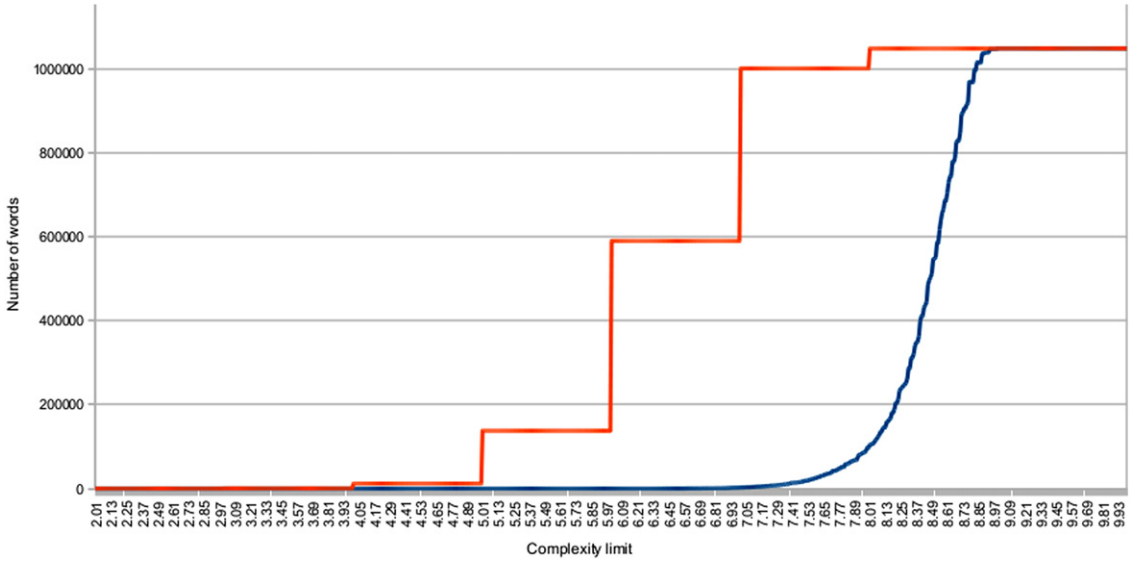

Fig. 1. For a two-symbol alphabet. Number of strings of length 20 up to given complexity. I-complexity in blue. L-complexity in orange.

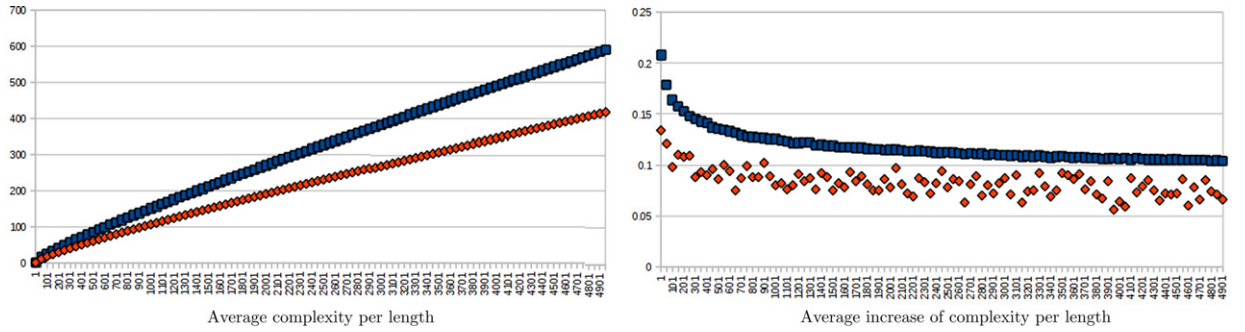

Fig. 2. For a two-symbol alphabet. On a Monte Carlo sample of strings up to length 5000.

Example 41. For any $c, d \in \mathcal{A}$, $c \neq d$ and $n \geq 2$, $L(c^n d) = 2$ and $L((c^n d)^r) = L(dc^n) = 3$.

This means the number of strings with L-complexity up to a given value is unbounded. Moreover, for any complexity greater than one, there are infinitely many strings with exactly that L-complexity.

Lemma 42. *For any integer $n \geq 2$, there exist infinitely many strings s such that $L(s) = n$.*

Proof. Consider any exhaustive production history of n steps. Changing the length parameter of the last production yields another production history that is still exhaustive, and changes the length of the produced string. Thus, different values produce different strings, all having an exhaustive production history of n steps, and by Theorem 40, an L-complexity of n. □

Given the above result, it is impossible to consider the Lempel–Ziv complexity a measure of the number of symbols needed to encode a string. By Theorem 39, our I-complexity does not give such a measure either. However, by Corollary 31, the number of symbols needed to encode a string is bounded by a quadratic function of its I-complexity. So, up to this extent, the I-complexity can be considered a measure of an encoding size.

4.2. Lempel–Ziv complexity and I-complexity are close

Both functions, L and I, are subadditive for concatenation, monotone in the prefix ordering on strings, and satisfy that most strings have almost maximal complexity. In spite of their very different behavior on infinitely many strings, they are close to each other.

Figs. 1 and 2 compare the distributions of the I-complexity (in blue) and of the L-complexity (in orange), for an alphabet of size 2. Fig. 1 considers all strings of length 20 and depicts the number of strings that have I-complexity up to a given value. The shape in Fig. 1 shows that the peak of increase in the number of strings of a given I-complexity is not exactly at the maximum value, but closely below it, which accurately reflects the theoretical results of Section 3. The two graphs in Fig. 2 use the same Monte Carlo sample of 1, 000, 000 strings of length up to 5000. The left graph depicts their average

complexity per length. The right graph shows the average increase in complexity obtained by adding a symbol (i.e., the discrete derivative of the previous function). The figures plot a similar behavior of the two complexity functions, though the I-complexity is smoother. The I-complexity seems to majorize the L-complexity. While this is not a universal truth, for instance, $L(cdc) = 2 < 2 + \mathrm{dlog}_2\, 2 = I(cdc)$, the data supports that it may hold for sufficiently long strings. The next result bounds the value of I in terms of L.

Theorem 43. *For every string s, $L(s)/(2\log_2 \alpha) \leq I(s) \leq L(s)\log_\alpha(|s|+1)$.*

Proof. The first inequality follows immediately from Lemma 29 and Theorem 40. For the second, the proof is similar to the one of Lemma 29, and uses the same notation. Let $n = L(s)$. For $1 \leq i \leq n$, let p_i be the first symbol produced by production i in the exhaustive production history of s. For completion, let $p_{n+1} = |s|+1$. Let $\ell_i = p_{i+1} - p_i$ be the length of each production, and set $\ell_0 = 0$. By definition of production, the substring $s[p_i..p_{i+1} - 2]$ is repeated before in the string. This means that for every $1 \leq i \leq n$, $B_s[p_{i+1} - 2] \geq \ell_i - 1$. Applying Observation 5.4 we can see that for each j, $B_s[p_{i+1} - 2 - j] \geq \ell_i - 1 - j$, which implies $[\![1, \ell_i - 1]\!]0 \preceq B_s[p_i..p_{i+1} - 1]$. Thus,

$$I(s) = I(B_s) = \sum_{i=1}^{n} I(B_s[p_i..p_{i+1} - 1]) \leq \sum_{i=1}^{n} I([\![1, \ell_i - 1]\!]0) = \sum_{i=1}^{n} I([\![0, \ell_i - 1]\!])$$
$$= \sum_{i=1}^{n} \log_\alpha(\ell_i + 1) \leq n \log_\alpha(|s|+1). \quad \square$$

The question of whether the I-complexity is strictly higher than L-complexity for sufficiently long strings remains open.

5. Open problems and ongoing work on the I-complexity

The study of the I-complexity opens a number of research venues:

Problem 1. Posed by Eugene Asarin[1]: Develop the I-complexity for timed sequences to obtain an efficient alternative to the computationally expensive notion of volume of regular timed languages [2,3].

Problem 2. Define a family of low I-complexity strings and study their properties. While I-complexity seems well suited for this, compressors and Lempel–Ziv complexity are not.

Problem 3. Sorted arrays form a distributive lattice. Investigate the algebraic structure of domination relation on sorted repetitions arrays.

Problem 4. Study the probability distribution of the I-complexity on the set of strings.

Problem 5 (*Technical*)**.** Improve the upper bound of the number of strings having I-complexity up to a given value, given in Corollary 31.

Problem 6. Develop I as an entropy. Although the I-complexity is not the length of a one-to-one encoding of strings, it is possible to give an entropy interpretation of I based on the upper bound on the number of strings with I-complexity up to a given value.

Problem 7. Study the theory of the conditional I-complexity and the notion of distance.

Problem 8. Investigate the properties of the I-complexity on infinite sequences; in particular, I-triviality, I-randomness and normality.

Acknowledgments

We thank Eugene Asarin for his insights on the I-complexity and for posing Problem 5.1. We also thank Alexander Shen, Laurent Bienvenu and an anonymous referee for their comments. This research has been supported by Agencia Nacional de Promoción Científica y Tecnológica and Universidad de Buenos Aires.

References

[1] Donald Adjeroh, Tim Bell, Amar Mukherjee, The Burrows-Wheeler Transform: Data Compression, Suffix Arrays, and Pattern Matching, Springer, 2008.
[2] E. Asarin, A. Degorre, Volume and entropy of regular timed languages: Analytic approach, in: FORMATS'09, in: LNCS, vol. 5813, Springer-Verlag, 2009, pp. 13–27.
[3] E. Asarin, A. Degorre, Volume and entropy of regular timed languages: Discretization approach, in: CONCUR'09, in: LNCS, vol. 5710, Springer-Verlag, 2009, pp. 69–83.

[1] Personal communication.

[4] Verónica Becher, Pablo A. Heiber, On extending de Bruijn sequences, Information Processing Letters 111 (2011) 930–932.
[5] N.G. Bruijn, A combinatorial problem, Koninklijke Nederlandse Akademie v. Wetenschappen 49 (1946) 758–764.
[6] Harry Buhrman, Lance Fortnow, Sophie Laplante, Resource-bounded Kolmogorov complexity revisited, in: Proceedings of the 14th Symposium on Theoretical Aspects of Computer Science, Springer, 1997, pp. 105–116.
[7] Cristian Calude, Kai Salomaa, Tania Roblot, Finite-state complexity and the size of transducers, Proceedings of Descriptional Complexity of Formal Systems (DCFS) (2010) 38–47.
[8] Gregory J. Chaitin, A theory of program size formally identical to information theory, Journal of the ACM 22 (1975) 329–340.
[9] Rudi Cilibrasi, Paul Vitanyi, Clustering by compression, IEEE Transactions on Information Theory 51 (4) (2005) 1523–1545.
[10] Andrzej Ehrenfeucht, Jan Mycielski, A pseudorandom sequence: how random is it? in Richard K. Guy, Unsolved problems, American Mathematical Monthly (1992) 373–375.
[11] Toru Kasai, Gunho Lee, Hiroki Arimura, Setsuo Arikawa, Kunsoo Park, Linear-time longest-common-prefix computation in suffix arrays and its applications, in: Proceedings of the 12th Annual Symposium on Combinatorial Pattern Matching, Springer-Verlag, 2001, pp. 181–192.
[12] A.N. Kolmogorov, Three approaches to the quantitative definition of information, Problems of Information Transmission 1 (1) (1965) 1–7.
[13] A. Lempel, J. Ziv, On the complexity of finite sequences, IEEE Transactions on Information Theory 22 (1) (1976) 75–81.
[14] María López-Valdés, Lempel–Ziv dimension for Lempel-Ziv compression, in: MFCS, 2006, pp. 693–703.
[15] Udi Manber, Gene Myers, Suffix arrays: a new method for on-line string searches, SIAM Journal on Computing 22 (5) (1993) 935–948.
[16] C.P. Schnorr, Process complexity and effective random tests, Journal of Computer Systems Science 7 (1973) 376–388.
[17] Jeffrey Shallit, Ming-Wei Wang, Automatic complexity of strings, Journal of Automata, Languages and Combinatorics 6 (4) (2001) 537–554.
[18] C.E. Shannon, A mathematical theory of communication, Bell System Technical Journal 27 (1948) 379–423. 623–656.
[19] V.A. Uspensky, A.Kh. Shen, Relations between varieties of Kolmogorov complexities, Mathematics Systems Theory 29 (1996) 271–292.
[20] Paul Vitányi, Ming Li, An Introduction to Kolmogorov Complexity and Its Applications, 2nd edition, Springer, 1997.
[21] Jacob Ziv, Abraham Lempel, Compression of individual sequences via variable-rate coding, IEEE Transactions on Information Theory 24 (5) (1978) 530–536.
[22] A.K. Zvonkin, L.A. Levin, The complexity of finite objects and the development of the concepts of information and randomness by means of the theory of algorithms, Russian Mathematical Surveys 25 (1970) 83–124.