Entropy Lower Bounds for Dictionary Compression

Michał Gańczorz
Institute of Computer Science, University of Wrocław, Poland
mga@cs.uni.wroc.pl

Abstract

We show that a wide class of dictionary compression methods (including LZ77, LZ78, grammar compressors as well as parsing-based structures) require $|S|H_k(S) + \Omega(|S|k\log\sigma/\log_\sigma|S|)$ bits to encode their output. This matches known upper bounds and improves the information-theoretic lower bound of $|S|H_k(S)$. To this end, we abstract the crucial properties of parsings created by those methods, construct a certain family of strings and analyze the parsings of those strings. We also show that for $k = \alpha\log_\sigma|S|$, where $0 < \alpha < 1$ is a constant, the aforementioned methods produce an output of size at least $\frac{1}{1-\alpha}|S|H_k(S)$ bits. Thus our results separate dictionary compressors from context-based one (such as PPM) and BWT-based ones, as the those include methods achieving $|S|H_k(S) + \mathcal{O}(\sigma^k\log\sigma)$ bits, i.e. the redundancy depends on k and σ but not on $|S|$.

2012 ACM Subject Classification Theory of computation → Data compression

Keywords and phrases compression, empirical entropy, parsing, lower bounds

Digital Object Identifier 10.4230/LIPIcs.CPM.2019.11

Funding Supported under National Science Centre, Poland project number 2017/26/E/ST6/00191.

1 Introduction

Dictionary compression. Dictionary compression is one of the most known and intensively studied area in data compression. As a result, there are many both simple and efficient methods that are commonly used in practice, those include algorithms from Lempel-Ziv family: LZ77 [34] and LZ78 [35], grammar compressors: Re-Pair [23], Greedy [4, 5], Sequitur [28], Sequential [33] as well as others [20, 32]. On top of that, there are also compressed data structures based on dictionary compression [16, 13, 8, 3, 22, 25, 7, 8, 19, 26].

Many methods mentioned above share a common feature – they induce a *parsing* of the input string. This is explicit for LZ77 and LZ78, as they both output a series of phrases, for grammar compressors the parsing is not always explicit, but it can be done fairly easy. For the mentioned data structures this is also either explicit [16, 13, 3, 22, 25] or easily done [7, 8]. The actual encodings of the phrases differ between the methods, but often we can lower bound their size by (zeroth order) entropy of the induced parsing. For example, in LZ78, for parsing with c phrases each phrase is assigned different (prefix-free) bit code, thus (by the properties of the prefix-free codes) this claim trivially holds (note also that all phrases in parsing induced by LZ78 are different). Another good example is grammar compressors: Re-Pair explicitly encodes the starting string of the grammar using entropy coder [23], Sequential uses its own encoding [33], which can be lower bounded by zeroth order entropy, etc.

Higher order empirical entropy. The k-th order empirical entropy, denoted $H_k(S)$ for a string S, is one of the most widely used measure of compressibility of texts, as on one hand in practice it is a good estimation of the "compressibility" of the text and on the other hand there are compressors that roughly achieve it. Surprisingly, for many dictionary compression methods it was shown that on a text S over an alphabet of size σ their output

30th Annual Symposium on Combinatorial Pattern Matching (CPM 2019).
Editors: Nadia Pisanti and Solon P. Pissis; Article No. 11; pp. 11:1–11:18
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

is of size at most

$$\beta|S|H_k(S) + \mathcal{O}(k|S|\log\sigma/\log_\sigma|S|) + \mathcal{O}(|S|\log\log_\sigma|S|/\log_\sigma|S|) \tag{1}$$

bits, for some constant β (ideally $\beta = 1$) [16, 13, 21, 27, 15, 30]. When $\beta = 1$ then the terms other than $|S|H_k(S)$ are often called the *redundancy* of the encoding. The summands under $\mathcal{O}$ notation are in $o(|S|\log\sigma)$ for $k = o(\log_\sigma|S|)$, such term is often treated as "lower order term". To capture this phenomenon, Kosaraju and Manzini introduced the notion of *coarse optimality* [21]: an algorithm is (β) coarse optimal if it achieves $\beta|S|H_k(S) + o(|S|\log\sigma)$ bits (our definition is slightly different than the original one, in which it was assumed that k and σ are constant). However, for larger k the additional term in (1) is in $\Omega(|S|\log\sigma)$, which is not satisfactory, as $|S|H_k(S) \leq |S|\log\sigma$ and so the additional term dominates the main one.

The bound in (1) for particular methods is in fact connected to a recently proved general upper bound on (zeroth-order) entropy of *arbitrary* parsing:

▶ **Theorem 1** ([15, Theorem 7]). *For any parsing $Y_S = y_1y_2\cdots y_c$ of S:*

$$cH_0(Y_S) \leq |S|H_k(S) + ck\log\sigma + |L|H_0(L) \ , \tag{2}$$

where L is string whose letters are lengths of consecutive phrases in Y_S, i.e. $L = |y_1|\,|y_2|\cdots|y_c|$.

To see the connection, observe that parsings produced by considered methods are usually of size at most $\mathcal{O}(|S|/\log_\sigma|S|)$, thus $ck\log\sigma$ turns to $\mathcal{O}(k|S|\log\sigma/\log_\sigma|S|)$, and for such sizes the $|L|H_0(L)$ can be estimated by $\mathcal{O}(|S|\log\log_\sigma|S|/\log_\sigma|S|)$. Of course, the considered methods encode also other information, which increases the constants in $\mathcal{O}$ notation or increases the constant in front of $|S|H_k(S)$.

Our results. The main result of this paper is a proof that the estimation (1) is tight for $k < \log_\sigma|S|$, at least for "natural" algorithms. Since estimating the size of the produced output is hard due to different encodings, we prove that the estimation on parsing entropy given by Theorem 1 is tight for such "natural" algorithms; note that this means that we disregard the size of other parts of the output. This is the reason, why we have to restrict the considered parsings, and so also the algorithms producing them, as it is impossible to, say, show any lower bound for a trivial parsing which consists of one phrase. To this end we define the class of *natural parsers*; this definition covers most of the dictionary based compressors, in particular it contains LZ77, LZ78, Re-Pair and grammar compressors producing irreducible grammars (such as Greedy or Sequitur). It also covers succinct data structures which parse the string into short phrases. We require that each phrase $y = wa$ in a parsing of the input string S induced by an algorithm is short (i.e. $|y| < \log_\sigma|S|$) or w occurs in S more than once. Note that the seemingly artificial condition on w and not on $y = wa$ is used in order to capture LZ77 and LZ78; the condition on short phrases ensures that some structures based on parsing into equal-length blocks [16, 13] are also covered.

Theorem 1 holds for any parsing, and it is known that the bound can be made more subtle when we can choose a specific parsing: at least one of l trivial parsings into equal-length phrases of length l achieves the mean of the first l entropies (plus smaller order term).

▶ **Theorem 2** ([15, Theorem 2]). *Let S be a string over an alphabet of size σ. Then for any integer l we can construct a parsing Y_S of size $|Y_S| \leq \left\lceil \frac{|S|}{l} \right\rceil + 1$ satisfying:*

$$|Y_S|H_0(Y_S) \leq |S|\frac{\sum_{i=0}^{l-1} H_i(S)}{l} + \mathcal{O}(\log|S|) \ .$$

All phrases except the first and last one have length l.

It is easy to give examples, for which Theorem 2 is tight, for instance de Bruijn sequence for $l \leq \log_\sigma |S|$ are examples of Theorem 2 tightness. On the other hand, natural parsers can perform better on de Bruijn strings than the bound in (1): we can show that they achieve $|S|H_k(S) + \mathcal{O}(|S| \log \log_\sigma |S| / \log_\sigma |S|) = |S| \log \sigma + \mathcal{O}(|S| \log \log_\sigma |S| / \log_\sigma |S|)$ for $k < \log_\sigma |S|$. This is because we can assign to each factor of length l a prefix-free code of length $\log \log_\sigma |S| + l \log \sigma$ bits.

We construct nontrivial examples on which bound from Theorem 2 is tight for any parsing produced by natural parsers for $l \in \Theta(\log_\sigma |S|)$. Moreover, each constructed string S satisfies

$$\log \sigma = H_0(S) \approx H_1(S) \ldots \approx H_{k-1}(S) \approx c \cdot H_k(S) \approx \ldots \approx c \cdot H_{l-1}(S) \ , \qquad (3)$$

where k can be any number such that $k \leq \log_\sigma |S|$, c is some constant greater than 1, and $\approx$ means that any two values from the sequence differ by at most $\mathcal{O}(\text{polylog}|S|/|S|) = o(1)$. In particular, in the limit for $|S| \to \infty$ they all tend to the same value. Those strings demonstrate that the first $\mathcal{O}(\cdot)$ summand in (1) is asymptotically optimal:

▶ **Corollary 3** (short version of Corollary 12). *Let* A *be a natural parser. Then for large enough σ there exists an infinite family of strings $\{S_n\}_{n=1}^{\infty}$ over σ-size alphabet such that the size of the output generated by* A *on $S \in \{S_n\}_{n=1}^{\infty}$ is at least $|S|H_k(S) + \Omega\left(\frac{|S| k \log \sigma}{\log_\sigma |S|}\right)$, for $k \leq \log_\sigma |S| - \frac{1}{2}$*

We can generalize the construction so that c in (3) is arbitrarily large (it depends on l, though), as a result we are able to prove that for $k = \alpha \log_\sigma |S|$, where $0 < \alpha < 1$, natural parsers output cannot be bounded with respect to $H_k(S)$.

▶ **Corollary 4** (short version of Corollary 14). *Let $0 < \alpha < 1$ be a rational constant and* A *be a natural parser. Then for large enough σ there exists an infinite family of strings $\{S_n\}_{n \in \mathbb{N}}$ over σ-size alphabet such that if* A *achieves $\beta |S| H_k(S) + o(|S| \log \sigma)$ bits on each $S \in \{S_n\}_{n \in \mathbb{N}}$ for $k = \alpha \log_\sigma |S|$, then $\beta \geq \frac{1}{1-\alpha}$.*

The bounds provided in Corollary 3 and Corollary 4 give higher lower bounds than information-theoretic ones, i.e. those based on counting the number of strings satisfying certain conditions. In fact, our proofs do not use information-theoretic arguments, instead, we give explicit construction of a string and analyze how natural parsers can parse it.

Consequences. It was asked [27], whether one can bound the size of Re-Pair's output by $c|S|H_k(S) + o(|S| \log \sigma)$, for some constant c, when $k = \alpha \log_\sigma |S|$. Corollary 4 settles this question in the negative.

The provided bounds can be used to compare the parsing-based methods with context-based ones (like PPM [9]) or ones based on BWT [11, 24]. It is known that their output is of size at most $|S|H_k(S) + \mathcal{O}(\sigma^k \cdot \log \sigma)$, even for $k = \alpha \log_\sigma |S|$, i.e. their redundancy is $o(|S|)$ [11, 24, 10]. Thus our theoretical bound backs a practically observed phenomenon of superiority of those methods over dictionary-based ones, at least for non-repetitive texts [9, 2, 1].

Lastly, we believe that the strings constructed in this paper are interesting on their own and can become "benchmark strings" for data compression algorithms.

Value of k. In the paper we consider $k = o(\log_\sigma |S|)$ and $k = \alpha \log_\sigma |S|$ for a constant $0 < \alpha < 1$. Let us comment, why those are the reasonable values of k (as a function of $|S|$ and σ). The value of k cannot be too large: for $k \geq \log_\sigma |S|$ the H_k can be smaller than information-theoretic bound [14]. The case of "small" k is clearly $k = o(\log_\sigma |S|)$: on one side it is traditionally thought to be small, on the other Corollary 3 shows that it is the necessary assumption to obtain reasonable bounds for a wide range of compression algorithms. Lastly, the intermediate case of $k = \alpha \log_\sigma k$ for some constant $0 < \alpha < 1$ considered in Corollary 4 is also well-motivated: the already mention context or BWT-based methods have good theoretical bounds for such k.

Related results. Kosaraju and Manzini considered also stronger notions than mentioned above: an algorithm is λ-*optimal* if it gives output of size at most $\lambda|S|H_k(S)+o(|S|H_k(S))$ [21]. They showed many negative results for λ-optimality of LZ77 and LZ78: the LZ78 is not λ-optimal for any k, and that LZ77 is not λ-optimal for $k = 1$ (but is 8-optimal for $k = 0$). Still, we know very little about λ-optimality of compression algorithms and our results seem to only give partial negative answer for large enough k.

2 Strings and their parsings

A string is a sequence of elements, called *letters*, from a finite set, called *alphabet*, and it is denoted as $w = w_1 w_2 \cdots w_k$, where each w_i is a letter, its length $|w|$ is k; alphabet's size is denoted by σ, the alphabet is usually not named explicitly as it is clear from the context or not needed, Γ is used when some name is needed. For any two strings w, w' the ww' denotes their concatenation. By $w[i \mathinner{.\,.} j]$ we denote $w_i w_{i+1} \cdots w_j$, this is a *substring* of w; ϵ denotes the empty string. For a pair of strings v, w the $|w|_v$ denotes the number of different (possibly overlapping) substrings of w equal to v; if $v = \epsilon$ then we set $|w|_\epsilon = |w|$.

A *parsing* of a string S is any representation $S = y_1 y_2 \cdots y_c$, where each y_i is nonempty and is called a *phrase*. We denote a parsing as $Y_S = y_1, \ldots, y_c$ and treat it as a string of length c over the alphabet $\{y_1, \ldots, y_c\}$; in particular $|Y_S| = c$ is its size.

Given a string w its k-*order empirical entropy* is

$$H_k(w) = -\frac{1}{|w|} \sum_{\substack{v\colon |v|=k \\ a\colon \text{letter}}} |w|_{va} \log\left(\frac{|w|_{va}}{|w|_v}\right) ,$$

with the convention that the summand is 0 whenever $|w|_{va} = 0$. We are mostly interested in the H_k entropy of the input string S and in the $H_0(Y_S)$ for parsing Y_S of S. The former is a natural measure of the input, to which we shall compare the size of considered algorithms, and the latter is a space lower bound for those algorithms, see Definition 5.

3 Natural parsers

We define the class of *natural parsers*. The general idea is that phrases generated by natural parsers are either short or occur at least twice in the string. Those are natural heuristics and so natural parsers include many practically used algorithms.

▶ **Definition 5.** *An algorithm is a* natural-parser *if given a string S over an alphabet of size σ it produces its parsing Y_S such that for each phrase $y = wa$, $|a| = 1$ of Y_S either $|S|_w > 1$, or $y \leq \log_\sigma |S|$; moreover, it encodes Y_S using at least $|Y_S|H_0(Y_S)$ bits.*

The condition that for a phrase $y = wa$ we require $|S|_w > 1$ instead of $|S|_y > 1$ was added so that natural parsers include also LZ78 and non self-referencing LZ77. Note that phrases of length 1 occurring once are allowed, as for them $w = \epsilon$ and $|S|_\epsilon = |S| > 1$.

Natural parsers include: Re-Pair, algorithms producing irreducible grammars, LZ78, LZ77 (for some natural encodings) and compressed text representations that partition the text into short (i.e. $\Theta(\log_\sigma |S|)$) blocks and encode the blocks using zeroth-order entropy [16, 13]. We now argue that all of the above algorithms are natural parsers.

LZ77 and LZ78. The LZ77 [34] and LZ78 [35] are probably the most known dictionary compressors. Each of them processes the input from left to right and builds a parsing in the process. In the LZ77 if, at some point, we processed some prefix P (of length j) of S the next phrase will be the longest possible substring starting at $S[j+1]$ which occurs in S at position at most j, plus one letter. We distinguish between *self-referencing* LZ77, where the previous occurrence of string of a factor may overlap the factor, and *non-self-referencing*, where it cannot (in other words, it occurs in P). Observe that both definition satisfy the first condition of natural parsers. When it comes to the entropy condition, the original encoding of LZ77 stored for each phrase its starting position in P using fixed length code, so it consumed at least $\log |S|$ bits per phrase, in total this is at least the entropy of the parsing (recall that we have $H_0(S) \leq \log |S|$ for every S). Other encodings are also possible, they are more of a practical than theoretical improvement: for example Kosaraju and Manzini [21] stored the position using $\log |P|$ bits, which can be shown to be equivalent to the original one minus lower order term of $o(|S| \log \sigma)$. Sometimes we store the offset, i.e. the difference between the starting position of phrase in P and j, though on average none of this encodings give better bound than $\log |Y_S|$ bits per phrase. On top of that there are also some LZ77-based data structures, like LZ-indices [26, 19, 22], they also use at least $\log |Y_S|$ bits per phrase.

In the case of LZ78, we build a dictionary (at the beginning it is empty), when we processed some prefix of length j the next phrase is wa where w is the longest string which starts in $S[j]$ and is in the dictionary. After each step we update our dictionary and add newly created phrase to it. Thus the description of phrase consists of index of string in a dictionary and a letter. Even though this descriptions can be encoded differently, usually the encodings (including the original one [35]) assign different (prefix-free) bit-codes to different phrases, thus by optimality of Huffman coding $|Y_S| H_0(Y_S)$ is the lower bound on output size. Note also that no two phrases have the same description, so it is sufficient (and necessary) to assign each phrase a fixed code of $\log |Y_S|$ bits, in fact some LZ78-based methods work this way [31]. Note though, that in some variants with limited dictionary size this may not be true.

Re-Pair. Re-Pair [23] is a grammar compressor, which builds a grammar generating the input string in the following way: we start with input string S and iteratively replace occurrences of the most frequent digram AB in S with a new symbol X, adding a rule $X \to AB$ to grammar. We iterate this procedure as long as there is a digram occurring at least twice. At the end we are left with a string which naturally induces a parsing of S, the original encoding of Re-Pair uses zeroth-order entropy to encode the parsing. Note that due to the fact that we only replace digrams which occur twice, each phrase in the parsing has at least two occurrences in S.

Irreducible Grammars. There is a large family of grammar compressors generating *irreducible grammars*. For example this family includes Greedy [4, 5], Sequential [33], Sequitur [29] and LongestMatch [20]. The conditions for a grammar to be irreducible are that:

- each nonterminal that occurs on right-hand side of grammars productions must occur at least twice
- no pair of nonterminals occurs twice on the right-hand side of the grammars productions
- no two nonterminals expands to the same string.

As an example $\{S \to AABB; A \to ab; B \to cd\}$ is an irreducible grammar, but neither $\{S \to AAB; A \to ab; B \to cd\}$ nor $\{S \to AAAA; A \to ab\}$ are.

The parsing induced by the right-hand side of starting symbol already satisfies the condition of a natural parser. There are many methods of encoding such grammars. The simplest involves entropy-coding of grammars right-hand sides, other assignes prefix-free codes to nonterminals [33, 4, 29]. A more sophisticated method was proposed by Kieffer and Yang [20], they showed that we can obtain the parsing from irreducible grammars by subsequently substituting one occurrence of each nonterminal till there are none left (at the end we are left with one string consisting of nonterminals) and then apply entropy coder. Such methods still are natural parsers, as each nonterminal of an original grammar still occurs in the obtained string (by the property that each nonterminal occurs twice) and thus each phrase of a parsing has at least two occurrences in the input string.

Other examples. There are compressed text representations which partition the string into short (i.e. of length at most $\log_\sigma |S|$) blocks and encode the blocks using zeroth-order entropy coder [16, 13, 12]; clearly they are also natural parsers.

4 Lower bound on parsing-based methods

In this section we construct a family of strings for which the bounds from Theorem 2 and Theorem 1 are tight for natural parsers. Furthermore, for those strings the $H_0, \ldots, H_{k-1}$ are by a constant factor c larger than $H_k, \ldots, H_l$, for adequate $l \in \Theta(\log_\sigma |S|)$. For ease of presentation we first show the construction for $c = 2$ and next we generalize to arbitrarily large c. Note that $H_0 = cH_k$ means that the mean of entropies cannot be contained in lower order term such as $o(|S| \log \sigma)$.

Our construction extends the one of de Bruijn strings, which, for a given alphabet Γ and order k, contain exactly once each string $w \in \Gamma^k$ as a substring; de Bruijn strings were used for proving lower bounds on compressibility before [14]; yet, contrary to previous results, our lower bounds are not information-theoretic.

▶ **Theorem 6.** *For every $k > 0$, $l \geq 0$, $p \geq 1$ there exists a string S over alphabet of size $\sigma = 4^p$ of length $\sigma^{k+\frac{l+1}{2}}$ such that:*

1. $\log \sigma - \mathcal{O}\left(\frac{i \log |S|}{|S|}\right) \leq H_i(S) \leq \log \sigma$ *for $i < k$;*
2. $\frac{\log \sigma}{2} - \mathcal{O}\left(\frac{i \log |S|}{|S|}\right) \leq H_i(S) \leq \frac{\log \sigma}{2}$ *for $k \leq i \leq k + l$;*
3. *no string of length $k + l + 1$ occurs more than once in S.*

For $l = 0$ the promised family are constructed from de Bruijn strings by appropriate letter merges. For de Bruijn strings the frequency of each substring depends (almost) only on its length, thus the bounds for the entropy of strings constructed in this way are easy to show. For larger l we make an inductive (on l) construction, similar in spirit to construction of de Bruijn strings: we construct a graph with edges labelled with letters and the desired string corresponds to an Eulerian cycle in this graph; to be more precise, the $(l+1)$st graph is exactly the line graph of the lth one. We guarantee that the frequency of strings depends only on their lengths, the exact condition is more involved than in case of de Bruijn strings.

The promised family of strings is much easier to define if we think of them as *cyclic strings*, meaning that after reading the last letter we can continue to read from the beginning. To distinguish strings from cyclic strings we denote by $S^{\circlearrowleft}$ the cyclic variant of S. Note that for a cyclic string $S^{\circlearrowleft}$ we are interested only in the occurrences of substrings in it (as defined below) thus the technical details of how many times we read S cyclically or when do we stop, are unimportant.

A string w occurs in $S^{\circlearrowleft}$ if w occurs in S, or $w = w_1 w_2$ and w_1 is a suffix of S and w_2 a prefix; we still require that $|w| \leq |S|$. The starting positions of an occurrence of w in $S^{\circlearrowleft}$ is defined naturally, two occurrences are different if they start at different positions. Denote by $|S|_w^{\circlearrowleft}$ the number of different occurrences of w in $S^{\circlearrowleft}$. Using this notation we define cyclic k-order entropy as:

$$H_k^{\circlearrowleft}(w) = -\frac{1}{|w|} \sum_{\substack{v:\, |v|=k \\ a:\text{ letter}}} |w|_{va}^{\circlearrowleft} \log \left(\frac{|w|_{va}^{\circlearrowleft}}{|w|_v^{\circlearrowleft}} \right) \quad .$$

The difference between cyclic and standard k-th order entropy is that it takes into the account also the first k letters of w. It is easy to show that it differs from $|w|H_k(w)$ in a small amount.

▶ **Lemma 7.** *For any string S and for any k we have:*

$$|S|H_k(S) + k \log |S| + \mathcal{O}(k) \geq |S|H_k^{\circlearrowleft}(S) \geq |S|H_k(S) \quad .$$

We now give the main construction, using the cyclic occurrences the estimation of Theorem 6 simplify as follows:

▶ **Lemma 8.** *For every $k > 0$, $l \geq 0$, $p \geq 1$ there exists a string S of length $\sigma^{k+\frac{l+1}{2}}$ over an alphabet Γ' of size $\sigma = 4^p$ such that:*
(dB1) *For every $w \in (\Gamma')^i, i < k$ we have $|S|_w^{\circlearrowleft} = \sigma^{k-i+(l+1)/2}$,*
(dB2) *For every $w \in (\Gamma')^i, k \leq i \leq k+l+1$ we have either $|S|_w^{\circlearrowleft} = \sigma^{(k+l+1-i)/2}$ or $|S|_w^{\circlearrowleft} = 0$,*
(dB3) *No string of length $k+l+1$ occurs cyclically more than once in S.*

Proof. Fix k, by S_l we will denote the string that satisfies the conditions (dB1–dB3) for l.

Let us first construct S_0. Consider cyclic de Bruijn sequence $B^{\circlearrowleft} = a_1 a_2 \cdots a_n$ of order $2k+1$ over an alphabet Γ of size $\sqrt{\sigma}$ (this is well defined, as $\sigma = 4^p$). Then $|B| = (\sqrt{\sigma})^{2k+1} = \sigma^{k+\frac{1}{2}}$. Consider two parsings of $B^{\circlearrowleft}$ into pairs of letters:

$$Y_B^1 = |a_1 a_2|a_3 a_4| \cdots |a_{n-3} a_{n-2}|a_{n-1} a_n| \quad Y_B^2 = |a_2 a_3|a_4 a_5| \cdots |a_{n-2} a_{n-1}|a_n a_1|$$

Now replace each pair a_i, a_j with a new symbol $b_{i,j}$, such that $b_{i,j} \neq b_{i',j'}$ if and only if $(a_i, a_j) \neq (a_{i'}, a_{j'})$. The size of the new alphabet Γ' is $\sigma = 4^p$. Consider the corresponding strings B_1' and B_2', treated in the following as cyclic strings:

$$B_1' = b_{1,2} b_{3,4} \cdots b_{n-3,n-2} b_{n-1,n} \qquad B_2' = b_{2,3} b_{4,5} \cdots b_{n-2,n-1} b_{n,1}$$

We can choose B such that it begins with a_i^{2k+1}, for some a_i. Then both strings B_1' and B_2' begin with $b_{i,i}^k$. Take $S_0 = B_1' B_2'$. Then, as the starting k-letters of both of them are the same, for each v of length at most $k+1$ it holds that

$$|B_1'|_v^{\circlearrowleft} + |B_2'|_v^{\circlearrowleft} = |S_0|_v^{\circlearrowleft} \quad .$$

We now calculate $|S_0|_w^{\circlearrowleft}$ for each possible w. For each k-letter string w' over Γ' the $|B_1'|_w^{\circlearrowleft} + |B_2'|_w^{\circlearrowleft}$ is $\sqrt{\sigma}$: w' is obtained from a fixed $2k$-letter string $w \in (\Gamma)^{2k}$ and such a string occurs cyclically $\sqrt{\sigma}$-times in $B^{\circlearrowleft}$, as there are $\sqrt{\sigma}$ ways to extend w to a $(2k+1)$-letter string and each such a string occurs cyclically exactly once in $B^{\circlearrowleft}$ and each cyclic occurrence of w in $B^{\circlearrowleft}$ yields one cyclic occurrence of w' in exactly one of $B_1'^{\circlearrowleft}$ and $B_2'^{\circlearrowleft}$. Moreover, as each string v of length $2k+1$ has exactly one occurrence in B, the letters after different cyclic occurrence of $w' \in (\Gamma')^k$ in $B_1'^{\circlearrowleft}$ or $B_2'^{\circlearrowleft}$ are pairwise different. Hence, each string of length at least $k+1$ over Γ' has at most one occurrence in S_0. For a string w of length $i<k$ observe that each of its σ^{k-i} extensions to a k-letter string occurs cyclically exactly $\sqrt{\sigma}$ times in $S_0^{\circlearrowleft}$, thus w occurs exactly $\sigma^{k-i+1/2}$. Thus S_0 satisfies conditions (dB1–dB3).

We now move to the general case of $l>0$. We cannot simply define S_l as a power of S_0, as then (dB3) is violated. Instead, we proceed similarly to the standard construction of de Bruijn strings: we will build a graph with vertices labelled with different strings of length $k+l+1$, define edges between strings that can be obtained by shifting by one letter to the right and show that this graph has a Hamiltonian cycle.

Define a family of directed graphs $G_0, G_1, \ldots$, where $G_i = (V_i, E_i)$. The nodes in V_i are labelled with (some) strings of length $k+1+i$ over Γ' (which is of size σ) and $E_i = \{(u,u') : \, u[2\ldots|u|] = u'[1\ldots|u'|-1]\}$. We label the edge from av to vb with avb. In case of G_0 its vertices V_0 are all cyclic substrings of $S_0^{\circlearrowleft}$ of length $k+1$. Recall that given a directed graph G its *line graph* $L(G)$ has edges of G as nodes and there is an edge (e,f) in $L(G)$ if and only if the end of e is the beginning of f. Define $G_{i+1} = L(G_i)$, observe that edges of G_i have labels that are strings of length $k+i+2$, those labels are reused as labels of nodes in G_{i+1}.

Let us state some basic properties of the defined graph: firstly, G_0 has in-degree and out-degree equal to $\sqrt{\sigma}$ (so it is $\sqrt{\sigma}$-regular): Given a node with a $k+1$-letter label w all its outgoing labels correspond to occurrences of the k letter suffix of w. And by (dB1) each k letter string has $\sqrt{\sigma}$ cyclic occurrences in $|S_0|^{\circlearrowleft}$ and each $k+1$ letter string has at most 1. So there are $\sqrt{\sigma}$ outgoing edges, each leading to a different node. Similar argument applies to the incoming edges. It is a folklore knowledge (and easy to show) that if G is d-regular then so is $L(G)$, moreover, if G is connected then so is $L(G)$; clearly G_0 is connected, as S_0 corresponds to a Hamiltonian path in it. Thus, all G_i's are Eulerian. It is well-known and easy to see that an Eulerian cycle in G corresponds to a Hamiltonian cycle in $L(G)$, thus each G_i has a Hamiltonian cycle.

We define the string $S_i^{\circlearrowleft}$ as the string read when traversing a Hamiltonian path in G_i (note that there may be many such paths: choose one arbitrarily): we begin with an arbitrary vertex u_0 in G_i, write its label and when we traverse the edge avb (so from av to vb) then we append b to the string. By easy induction we can show that when we are at a node labelled with u then the current string has u as a suffix. In particular, after traversing the whole path begins and ends with u_0. By identifying those two copies of u_0 we obtain a cyclic string. Note that a string w of length $k+i+1$ occurs at position p if and only if p-th vertex on the path is labelled with w. Concerning the length $|S_i|$, this is exactly $|V_i| = |E_{i-1}| = \sqrt{\sigma}|V_{i-1}|$, as each G_i is $\sqrt{\sigma}$ regular. Since $|V_0| = \sigma^{k+\frac{1}{2}}$, we conclude that $|V_i| = \sigma^{k+\frac{i+1}{2}}$. We also show that each occurrence of a string w of length $k+i$ in S_i is followed by a different letter, in particular this implies that a string w' of length $k+i+1$ has at most one occurrence in S_i, i.e. (dB3). We know that this is true for G_0, we proceed by induction. Consider all nodes labelled with wa for some letter a in G_{i+1}, where $|w| = k+i+1$. They all correspond to edges in G_i labelled with the same strings. Those edges originate from nodes labelled with w and as $|w| = k+i+1$, by induction assumption there is exactly one such node. Now, if there were two edges outgoing edge from w labelled with wa then they would lead to two vertices labelled with the same label w' (where $w' = w[2\mathinner{..}|w|]a$), which is not the case by the induction assumption. Hence, wa has at most one occurrence.

It is left to show that S_l satisfies (dB1)–(dB2). We proceed by induction on l: first we show that for any string w of length at most $k+l+1$ it holds that $|S_{l+1}|_w^{\circlearrowleft} = |S_l|_w^{\circlearrowleft} \cdot \sqrt{\sigma}$: observe that $|S_{l+1}|_w^{\circlearrowleft}$ is the number of nodes in G_{l+1} that have w as their prefix. This is exactly the number of edge-labels in G_l that have w as their prefix. But those labels are of length $k+l+2 > |w|$, thus edge $e = (u, u')$ has w as the prefix of its label if and only if this labels is also a prefix of label of u. As G_l is $\sqrt{\sigma}$-regular, each u is counted $\sqrt{\sigma}$ times (once for each of the $\sqrt{\sigma}$ outgoing edges) and so we obtain the claim.

It is left to consider the case when $|w| = k+l+2$, but strings of this lengths are labels of vertices of G_{l+1} and if w is label of some vertex of G_{l+1} then clearly $|S_{l+1}|_w = 1$. ◀

It is worth noting that the construction suggests that for a fixed k (and de Bruijn string) there are exponentially many (in l) strings satisfying conditions (dB1–dB3). Moreover the construction suggests that there are exponentially many (in k) such strings: it seems that for each de Bruijn string the constructed strings is different, and there are exponentially many de Bruijn strings of order k. Proving this does not seem easy and we leave it for future work.

▶ **Example 9.** For $\sigma = 4, k = 2, l = 0$: $S = aababcbbadccdbddaacadaccbdbbcddc$. Observe that each k-letter substring occurs cyclically $\sqrt{\sigma}$ times, the letters after those occurrences are pairwise different. $H_0^{\circlearrowleft}(S) = H_1^{\circlearrowleft}(S) = \log\sigma$ and $H_2^{\circlearrowleft}(S) = \frac{\log\sigma}{2}$.

For $\sigma = 4, k = 1, l = 1$ the string is $S = abbbdacdcacabdcd$ and $H_0^{\circlearrowleft}(S) = \log\sigma$, $H_1^{\circlearrowleft}(S) = H_2^{\circlearrowleft}(S) = \frac{\log\sigma}{2}$.

Proof of Theorem 6. Take the string S from Lemma 8 for a given k, l, p. By definition of cyclic entropy $H_i^{\circlearrowleft}(S) = \log\sigma$ for $i < k$ and $H_i^{\circlearrowleft}(S) = \frac{\log\sigma}{2}$ for $k \le i \le k+l$, moreover no string of length $k+l+1$ occurs cyclically more than once in S. By Lemma 7, string S satisfies the conditions stated in the Theorem. ◀

In the following we estimate, how bad a natural parser performs on a string from Theorem 6. It is easy to see that for such a string for parameters k, l it cannot make a phrase longer than $k+l+1$, as by (dB3) they have at most one occurrence, so we first lower-bound the entropy of such parses.

▶ **Lemma 10.** *Let S be a string from Theorem 6 for parameters k and l, and let $z = k+l+1$. Then for every parsing $Y_S = y_1y_2\ldots y_{|Y_S|}$ of S such that $|y_i| \le z$ we have:*

$$|Y_S|H_0(Y_S) \ge \frac{|S|(z+k)}{2z}\log\sigma - |Y_S|\log\frac{|S|}{|Y_S|} \quad .$$

Proof. Let $m = |Y_S|$ and $n = |S|$. For a string w let l_w be the number of occurrences of w in Y_S. Clearly $l_w \le |S|_w^{\circlearrowleft}$ and by construction for any w such that $|S|_w^{\circlearrowleft} > 0$:

$$|S|_w^{\circlearrowleft} = \begin{cases} \frac{n}{\sigma^{|w|}} & \text{for } |w| \le k \\ \frac{n}{\sigma^{(|w|+k)/2}} & \text{for } k < |w| \le z \end{cases} \quad . \tag{4}$$

thus

- $l_w \le \frac{n}{\sigma^{|w|}}$, for $|w| \le k$;
- $l_w \le \frac{n}{\sigma^{(|w|+k)/2}}$, for $k < |w| \le z$.

Define:

$$m_1 = \sum_{w \in Y_S, |w| \le k} l_w \qquad m_2 = \sum_{w \in Y_S, |w| > k} l_w$$

$$n_1 = \sum_{w \in Y_S, |w| \le k} |w| l_w \qquad n_2 = \sum_{w \in Y_S, |w| > k} |w| l_w$$

Note that as each phrase in Y_s has length at most z, we have that

$$m_2 z \ge n_2 \tag{5}$$

Then

$$\begin{aligned}
|Y_S|H_0(Y_S) &= \sum_{w \in Y_S} l_w \log \frac{m}{l_w} \\
&= \sum_{|w| \le k} l_w \log \frac{m}{l_w} + \sum_{|w|>k} l_w \log \frac{m}{l_w} \\
&\ge \sum_{|w| \le k} l_w \log \frac{m\sigma^{|w|}}{n} + \sum_{|w|>k} l_w \log \frac{m\sigma^{(|w|+k)/2}}{n} \\
&= \sum_{w} l_w \log \frac{m}{n} + \sum_{|w| \le k} l_w \log \sigma^{|w|} + \sum_{|w|>k} l_w \log \sigma^{(|w|+k)/2} && \text{from (4)} \\
&= m \log \frac{m}{n} + \sum_{|w| \le k} l_w |w| \log \sigma + \sum_{|w|>k} \frac{l_w |w|}{2} \log \sigma + \sum_{|w|>k} \frac{l_w k}{2} \log \sigma \\
&= m \log \frac{m}{n} + n_1 \log \sigma + \frac{n_2}{2} \log \sigma + \frac{m_2 k}{2} \log \sigma \\
&\ge m \log \frac{m}{n} + n_1 \log \sigma + \frac{n_2}{2} \log \sigma + \frac{n_2 k}{2z} \log \sigma && \text{from (5)} \\
&\ge \frac{n}{2} \log \sigma + \frac{nk}{2z} \log \sigma + m \log \frac{m}{n} \\
&= \frac{n(z+k)}{2z} \log \sigma - m \log \frac{n}{m} \quad .
\end{aligned}$$

◀

Natural parsers on strings defined in Theorem 6 cannot do much better than the mean of entropies, which gives general bounds on algorithms inducing natural parsers.

▶ **Theorem 11.** *Let* A *be a natural parser. Let $k \ge 0$ be an integer function of $|S|$ and σ such that for every σ for infinitely many $|S|$ it holds that $k_{|S|,\sigma} \le \log_\sigma |S| - \frac{1}{2}$, where $k_{|S|,\sigma}$ denotes the value of k for $|S|$ and σ. Then for any natural $p > 0$ there exists an infinite family of strings $\{S_n\}_{n=1}^{\infty} \subseteq \Gamma^*$, where $|\Gamma| = 4^p$, such that the bit-size of output $\mathsf{A}(S)$ of* A *on $S \in \{S_n\}_{n=1}^{\infty}$ is at least:*

$$\mathsf{A}(S) \ge |S|H_k(S) + \frac{\rho|S|\log\sigma}{2} - \lambda|S| \ge (1+\rho)\,|S|H_k(S) - \lambda|S| \ ,$$

where $\rho = \frac{k}{2\log_\sigma |S| - k}$ and $\lambda < 0.54$. If the size of parsing induced by A *is $o(|S|)$ then:*

$$\mathsf{A}(S) \ge |S|H_k(S) + \frac{\rho|S|\log\sigma}{2} - o(|S|) \ge (1+\rho)\,|S|H_k(S) - o(|S|) \ .$$

Proof. Denote $n = |S|$ and $m = |Y_S|$. The proof is a simple application of Lemma 10. Fix alphabet Γ of size $\sigma = 4^p$. Take k such that $k_{|S|,\sigma} \le \log_\sigma |S| - \frac{1}{2}$, then $l = 2\log_\sigma |S| - 2k_{|S|,\sigma} - 1$ is non-negative; note that due to assumptions we can take arbitrarily large $|S|$. Then $k_{|S|,\sigma} + \frac{l+1}{2} = \log_\sigma |S|$. So it is possible to construct a string S (of length $n = |S|$) from Theorem 6, for parameters $k_{|S|,\sigma}, l, p$.

Let $Y_S = y_1 y_2 \cdots y_{|Y_S|}$ be a parsing of S induced by A. As A is a natural parser, we have that $|y_i| \le k + \frac{l+1}{2}$. We use Lemma 10 to lower bound the output of the algorithm:

$$\begin{aligned}
\mathsf{A}(S) &\geq |Y_S|H_0(Y_S) \\
&\geq \frac{|S|(2k+l+1)}{2(k+l+1)}\log\sigma - m\log\frac{n}{m} \\
&\geq |S|H_k(S) + \frac{|S|k}{2(k+l+1)}\log\sigma - m\log\frac{n}{m} & \text{, as } \frac{\log\sigma}{2} \geq H_k(S) \\
&\geq |S|H_k(S) + \frac{\rho|S|\log\sigma}{2} - m\log\frac{n}{m} \\
&\geq (1+\rho)|S|H_k(S) - m\log\frac{n}{m} \ .
\end{aligned}$$

The expression $m\log\frac{n}{m}$ is minimized when $m = n/e$, so we can bound it by $n\frac{\log e}{e} < 0.54n$, and by $o(n)$ if $m = o(n)$. Plugging those values to the above equation yields the claim. ◀

There are several consequences of Theorem 11 for natural parsers (the proofs of the Corollaries are in the Appendix). First, for $k \leq \log_\sigma|S| - \frac{1}{2}$ they cannot achieve better redundancy than $\mathcal{O}(|S|k\log\sigma/\log_\sigma|S|)$ bits. If $o(\log_\sigma|S|)$ is the best bound on k, then the redundancy of $o(|S|\log\sigma)$ is necessary.

▶ **Corollary 12.** *Let* A *be a natural parser. Then for large enough* σ *there exists an infinite family of strings* $\{S_n\}_{n\in\mathbb{N}}$ *over a* σ*-size alphabet such that for each* $S \in \{S_n\}_{n\in\mathbb{N}}$*, the size of the output generated by* A *on* S *is at least*

$$\mathsf{A}(S) \geq |S|H_k(S) + \Omega\left(\frac{|S|k\log\sigma}{\log_\sigma|S|}\right) \ ,$$

where $k \leq \log_\sigma|S| - \frac{1}{2}$ *can be any function of* $(|S|,\sigma)$*.*

Theorem 2 is tight in the sense that we cannot make the constant at the mean of entropies smaller than 1, even if we allow phrases of different lengths (not too large, though).

▶ **Corollary 13.** *Let* k *be an integer function of* $(|S|,\sigma)$ *such that* $k \leq \log_\sigma|S| - \frac{1}{2}$*. Then for large enough* σ *there exists an infinite family of strings* $\{S_n\}_{n\in\mathbb{N}}$ *over a* σ*-size alphabet such that for each* $S \in \{S_n\}_{n=1}^{\infty}$

$$2H_k(S) + \mathcal{O}\left(\frac{\log|S|}{|S|}\right) \geq H_0(S) \geq 2H_k(S)$$

and no parsing Y_S *with phrases of length at most* $j = 2\log_\sigma|S| - k$ *achieves*

$$|Y_S|H_0(Y_S) \leq (1-\epsilon)\frac{|S|}{j}\sum_{i=0}^{j-1}H_i(S) + o(|S|\log\sigma) \ ,$$

for $\epsilon > 0$*.*

Finally, we show that extending the bounds to $k = \alpha\log_\sigma n$ for a constant $0 < \alpha < 1$ implies that $|S|H_k(S)$ (without a constant coefficient) is not achievable. This gives partial (negative) answer to the question, whether we can prove optimality results for Re-Pair for $k = \alpha\log_\sigma|S|$ [27]. For a statement in full generality, we need to extend the construction from Theorem 6. For the constructed strings the ratio of $H_0(S)$ and $H_k(S)$ can be arbitrary large (at the cost of increasing l).

▶ **Corollary 14.** *Let $0 < \alpha < 1$ be a rational constant and* A *a natural parser. For large enough σ there exists an infinite family of strings $\{S_n\}_{n=1}^{\infty}$ over σ-size alphabet such that if* A *achieves $\beta|S|H_k(S) + o(|S|\log\sigma)$ bits on each $S \in \{S_n\}_{n=1}^{\infty}$ for $k = \alpha\log_\sigma|S|$, then $\beta \geq \frac{1}{1-\alpha}$.*

Substituting $k = \alpha\log_\sigma n$ to Corollary 12 already shows that an output of a natural parser is at least $\frac{2}{2-\alpha}|S|H_k(S)$ (see (7) in the Appendix for calculations). To show a bound with coefficient $\frac{1}{1-\alpha}$ we generalize the construction from Lemma 8.

Lemma 8 shows that there exist strings for which $H_l(S) = H_k(S) = \frac{1}{2}H_{k-1}(S) = \frac{1}{2}H_0(S)$, for every k, l, such that $k < \log_\sigma|S| \leq l$. When $k = \alpha\log_\sigma|S|$ (for a constant $0 < \alpha < 1$) this guarantees that the mean of l first entropies is larger by a constant factor than $H_k(S)$. The intuition is that if we could construct the strings such that $|S|H_k(S) = \frac{1}{r}|S|H_0(S)$, for arbitrarily large r, we would get that the mean of entropies can be arbitrarily large with respect to $H_k(S)$. This can be done, we show corresponding properties.

▶ **Lemma 15** (cf. Lemma 8)**.** *For every $k > 0$, $l \geq 0$, $p \geq 1, r \geq 2$ there exists a string S over alphabet Γ' of size $\sigma = (2^r)^p$ of length $\sigma^{k+\frac{l+1}{r}}$ such that:*

(dB1') *For any $w \in (\Gamma')^i, i < k$ we have $|S|_w^{\circlearrowleft} = \sigma^{k-i+(l+1)/r}$,*

(dB2') *For any $w \in (\Gamma')^i, k \leq i \leq k+l+1$ we have either $|S|_w^{\circlearrowleft} = \sigma^{(k+l+1-i)/r}$ or $|S|_w^{\circlearrowleft} = 0$,*

(dB3') *No string of length $k+l+1$ occurs cyclically more than once in S.*

For the construction from Lemma 15, an appropriate variant of Lemma 10 holds.

▶ **Lemma 16** (cf. Lemma 10)**.** *Let S be a string from Lemma 15 for parameters k, l, p and r, let $z = k+l+1$. Then for every parsing $Y_S = y_1y_2\cdots y_{|Y_S|}$ of S such that $|y_i| \leq z$ we have:*

$$|Y_S|H_0(Y_S) \geq \frac{|S|(z+(r-1)k)}{r\cdot z}\log\sigma - |Y_S|\log\frac{|S|}{|Y_S|} \quad .$$

Now we can state the generalized version of Theorem 6:

▶ **Theorem 17** (cf. Theorem 6)**.** *For every $k > 0$, $l \geq 0$, $p \geq 1$, $r \geq 2$ there exists a string S over alphabet of size $\sigma = (2^r)^p$ of length $\sigma^{k+\frac{l+1}{r}}$ such that:*

1. $\log\sigma - \mathcal{O}\left(\frac{i\log|S|}{|S|}\right) \leq H_i(S) \leq \log\sigma$ *for $i < k$;*
2. $\frac{\log\sigma}{r} - \mathcal{O}\left(\frac{i\log|S|}{|S|}\right) \leq H_i(S) \leq \frac{\log\sigma}{r}$ *for $k \leq i \leq k+l$;*
3. *no string of length $k+l+1$ occurs more than once in S.*

5 Conclusions and open problems

Conclusions. We have shown space lower bounds for a large class of parsing-based compression and parsing methods: they yield output greater than $|S|H_k(S)$ by at least $\Omega(|S|k\log\sigma/\log_\sigma n)$ additional bits, thus even for fixed k and σ this value grows with $|S|$. Moreover we have shown that if $k = \alpha\log_\sigma|S|$ then parsing-based methods produce output of size at least $1/(1-\alpha)|S|H_k(S)$. These bounds hold assuming that we encode the parsing using zeroth-order entropy or similar method. Those bounds are strictly higher than upper bounds for methods based on BWT or PPM.

Open problems. There are parsing based compressed text representations [17, 12], which allow for fast random access to letters and substring retrieval, achieving $|S|H_k(S)+\mathcal{O}(\sigma^k\log|S|) = |S|H_k(S) + o(|S|)$ for $k = \alpha\log_\sigma|S|$, where $\alpha < \frac{1}{8}$; the difference is that they encode the parsing using first order entropy coder. Still, they do not allow random access in constant time nor short (i.e. $\Theta(\log_\sigma|S|)$) substring retrieval in constant time; both operations are facilitated when zeroth-entropy coder is used [16, 13, 12]. More precisely, structures based on first-order entropy coders achieve $\mathcal{O}(\log|S|/\log\log|S|)$ time for such operations. Can we estimate time-space tradeoffs?

Kosaraju and Manzini [21] considered the notion of λ-optimality: an algorithm is λ *optimal* if it achieves $\lambda|S|H_k(S) + o(|S|H_k(S))$ bits for some constant λ. They showed [21] that LZ77 and the slight modification of LZ78 are λ optimal for $k = 0$, and that neither LZ77 nor LZ78 are for $k > 0$. They left an open question, whether there is an parsing-based algorithm which is λ-optimal for $k > 0$. Corollary 14 implies that no natural parser is λ-optimal for $k = \alpha \log_\sigma n$ for any constant α, which partially answers this question. Still, other cases, for instance: of constant k, remain open. This is interesting because our constructed strings have high-entropy, and previously low entropy strings where used in the context of λ-optimality [21]. A natural question is, whether the dichotomy between natural parsers and PPM methods holds also for low entropy strings?

Is any grammar compressor λ-optimal, even for $k = 0$? On one hand, there are examples of small entropy strings on which most grammar compressors perform badly [6, 18], still this does not apply to all of them, e.g. Greedy is an exception.

References

1 Test results for data from Canterbury Corpus. http://corpus.canterbury.ac.nz/details/cantrbry/RatioByRatio.html. Accessed: 2019-01-17.

2 Test results for data from Pizza & Chilli corpus. http://pizzachili.dcc.uchile.cl/texts.html. Accessed: 2019-01-17.

3 Anisa Al-Hafeedh, Maxime Crochemore, Lucian Ilie, Evguenia Kopylova, William F. Smyth, German Tischler, and Munina Yusufu. A comparison of index-based Lempel-Ziv LZ77 factorization algorithms. *ACM Comput. Surv.*, 45(1):5, 2012.

4 Alberto Apostolico and Stefano Lonardi. Some theory and practice of greedy off-line textual substitution. In *Data Compression Conference, 1998. DCC'98. Proceedings*, pages 119–128. IEEE, 1998.

5 Alberto Apostolico and Stefano Lonardi. Compression of Biological Sequences by Greedy Off-Line Textual Substitution. In *Proceedings of the Conference on Data Compression*, DCC '00, pages 143–152, Washington, DC, USA, 2000. IEEE Computer Society. URL: http://dl.acm.org/citation.cfm?id=789087.789789.

6 Moses Charikar, Eric Lehman, Ding Liu, Rina Panigrahy, Manoj Prabhakaran, Amit Sahai, and Abhi Shelat. The smallest grammar problem. *IEEE Trans. Information Theory*, 51(7):2554–2576, 2005. doi:10.1109/TIT.2005.850116.

7 Francisco Claude and Gonzalo Navarro. Self-Indexed Grammar-Based Compression. *Fundam. Inform.*, 111(3):313–337, 2011. doi:10.3233/FI-2011-565.

8 Francisco Claude and Gonzalo Navarro. Improved Grammar-Based Compressed Indexes. In *String Processing and Information Retrieval - 19th International Symposium, SPIRE 2012, Cartagena de Indias, Colombia, October 21-25, 2012. Proceedings*, pages 180–192, 2012. doi:10.1007/978-3-642-34109-0_19.

9 J. Cleary and I. Witten. Data Compression Using Adaptive Coding and Partial String Matching. *IEEE Transactions on Communications*, 32(4):396–402, April 1984. doi:10.1109/TCOM.1984.1096090.

10 Łukasz Dębowski. Is Natural Language a Perigraphic Process? The Theorem about Facts and Words Revisited. *Entropy*, 20(2):85, 2018. doi:10.3390/e20020085.

11 Paolo Ferragina and Giovanni Manzini. Compression Boosting in Optimal Linear Time Using the Burrows-Wheeler Transform. In *Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '04, pages 655–663, Philadelphia, PA, USA, 2004.

Society for Industrial and Applied Mathematics. URL: http://dl.acm.org/citation.cfm?id=982792.982892.

12 Paolo Ferragina, Giovanni Manzini, Veli Mäkinen, and Gonzalo Navarro. Compressed representations of sequences and full-text indexes. *ACM Trans. Algorithms*, 3(2):20, 2007. doi:10.1145/1240233.1240243.

13 Paolo Ferragina and Rossano Venturini. A simple storage scheme for strings achieving entropy bounds. *Theor. Comput. Sci.*, 372(1):115–121, 2007. doi:10.1016/j.tcs.2006.12.012.

14 Travis Gagie. Large alphabets and incompressibility. *Inf. Process. Lett.*, 99(6):246–251, 2006. doi:10.1016/j.ipl.2006.04.008.

15 Michał Gańczorz. Entropy bounds for grammar compression. *CoRR*, abs/1804.08547, 2018. arXiv:1804.08547.

16 Rodrigo González and Gonzalo Navarro. Statistical Encoding of Succinct Data Structures. In Moshe Lewenstein and Gabriel Valiente, editors, *Combinatorial Pattern Matching, 17th Annual Symposium, CPM 2006, Barcelona, Spain, July 5-7, 2006, Proceedings*, volume 4009 of *Lecture Notes in Computer Science*, pages 294–305. Springer, 2006. doi:10.1007/11780441_27.

17 Roberto Grossi, Rajeev Raman, Srinivasa Rao Satti, and Rossano Venturini. Dynamic Compressed Strings with Random Access. In *Automata, Languages, and Programming - 40th International Colloquium, ICALP 2013, Riga, Latvia, July 8-12, 2013, Proceedings, Part I*, pages 504–515. Springer, 2013. doi:10.1007/978-3-642-39206-1_43.

18 Danny Hucke, Artur Jeż, and Markus Lohrey. Approximation ratio of RePair. *CoRR*, abs/1703.06061, 2017.

19 Juha Kärkkäinen and Erkki Sutinen. Lempel-Ziv Index for q-Grams. *Algorithmica*, 21(1):137–154, 1998. doi:10.1007/PL00009205.

20 John C. Kieffer and En-Hui Yang. Grammar-based codes: A new class of universal lossless source codes. *IEEE Transactions on Information Theory*, 46(3):737–754, 2000. doi:10.1109/18.841160.

21 S. Rao Kosaraju and Giovanni Manzini. Compression of Low Entropy Strings with Lempel-Ziv Algorithms. *SIAM J. Comput.*, 29(3):893–911, 1999. doi:10.1137/S0097539797331105.

22 Sebastian Kreft and Gonzalo Navarro. Self-indexing Based on LZ77. In *Combinatorial Pattern Matching - 22nd Annual Symposium, CPM 2011, Palermo, Italy, June 27-29, 2011. Proceedings*, pages 41–54, 2011. doi:10.1007/978-3-642-21458-5_6.

23 N. Jesper Larsson and Alistair Moffat. Offline Dictionary-Based Compression. In *Data Compression Conference*, pages 296–305. IEEE Computer Society, 1999. doi:10.1109/DCC.1999.755679.

24 Veli Mäkinen and Gonzalo Navarro. Dynamic entropy-compressed sequences and full-text indexes. *ACM Trans. Algorithms*, 4(3):32:1–32:38, 2008. doi:10.1145/1367064.1367072.

25 Gonzalo Navarro. Indexing Text Using the Ziv-Lempel Trie. *J. of Discrete Algorithms*, 2(1):87–114, March 2004. doi:10.1016/S1570-8667(03)00066-2.

26 Gonzalo Navarro and Veli Mäkinen. Compressed Full-text Indexes. *ACM Comput. Surv.*, 39(1), April 2007. doi:10.1145/1216370.1216372.

27 Gonzalo Navarro and Luís M. S. Russo. Re-Pair Achieves High-Order Entropy. In *DCC*, page 537. IEEE Computer Society, 2008. doi:10.1109/DCC.2008.79.

28 Craig G Nevill-Manning and Ian H Witten. Compression and explanation using hierarchical grammars. *The Computer Journal*, 40(2 and 3):103–116, 1997.

29 Craig G. Nevill-Manning and Ian H. Witten. Identifying Hierarchical Structure in Sequences: A Linear-Time Algorithm. *J. Artif. Intell. Res. (JAIR)*, 7:67–82, 1997. doi:10.1613/jair.374.

30 C. Ochoa and G. Navarro. RePair and All Irreducible Grammars are Upper Bounded by High-Order Empirical Entropy. *IEEE Transactions on Information Theory*, 2019. to appear.

31 Kunihiko Sadakane and Roberto Grossi. Squeezing succinct data structures into entropy bounds. In *Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*, pages 1230–1239. Society for Industrial and Applied Mathematics, 2006.

32 Michal Vasinek and Jan Platos. Prediction and Evaluation of Zero Order Entropy Changes in Grammar-Based Codes. *Entropy*, 19(5):223, 2017. `doi:10.3390/e19050223`.

33 En-Hui Yang and John C. Kieffer. Efficient universal lossless data compression algorithms based on a greedy sequential grammar transform - Part one: Without context models. *IEEE Trans. Information Theory*, 46(3):755–777, 2000. `doi:10.1109/18.841161`.

34 Jacob Ziv and Abraham Lempel. A universal algorithm for sequential data compression. *IEEE Trans. Information Theory*, 23(3):337–343, 1977. `doi:10.1109/TIT.1977.1055714`.

35 Jacob Ziv and Abraham Lempel. Compression of individual sequences via variable-rate coding. *IEEE Trans. Information Theory*, 24(5):530–536, 1978. `doi:10.1109/TIT.1978.1055934`.

A Additional material for Section 4

Construction of Generalized de Bruijn strings

Proof of Lemma 7. Fix some string v, $|v| = k$. Consider the difference

$$\sum_{a\text{: letter}} |w|_{va}^{\circlearrowleft} \log\left(\frac{|w|_{va}^{\circlearrowleft}}{|w|_{v}^{\circlearrowleft}}\right) - \sum_{a\text{: letter}} |w|_{va} \log\left(\frac{|w|_{va}}{|w|_{v}}\right) \quad , \tag{6}$$

our goal is to estimate it when summed over all v of length k.

Define S_1, S_2 as the strings of letters that follow cyclic occurrences (standard occurrences) of v in w, formally for each letter a they should satisfy

$$|S_1|_a = |w|_{va}^{\circlearrowleft} \qquad |S_2|_a = |w|_{va} \quad ,$$

note that this implies that

$$|S_1| = |w|_{v}^{\circlearrowleft} \qquad |S_2| = |w|_{v} \quad .$$

Then left and right summands from (6) are equal to, respectively:

$$|S_1|H_0(S_1) = \sum_{a\text{: letter}} |w|_{va}^{\circlearrowleft} \log\left(\frac{|w|_{va}^{\circlearrowleft}}{|w|_{v}^{\circlearrowleft}}\right) \qquad |S_2|H_0(S_2) = \sum_{a\text{: letter}} |w|_{va} \log\left(\frac{|w|_{va}}{|w|_{v}}\right)$$

We first show that (6) is positive, which yields the second inequality of the Lemma.

Clearly $|w|_{va}^{\circlearrowleft} \geq |w|_{va}$, and so we can obtain S_2 from S_1 by removing and permuting letters, which cannot increase the entropy. Hence $|S_1|H_0(S_1) \geq |S_2|H_0(S_2)$, which yields that (6) is positive and so the second inequality of the lemma follows.

To upper bound the difference in (6) observe that S_1 is obtained by adding symbols to S_2 and the addition of one letter to a string of length at most $|S| - 1$ increases the entropy by at most $\log|S| + \beta$, for some constant β. Moreover, there are at most k such additions, when summing over all possible k-length contexts v. Thus the first inequality holds. ◀

Proof of Corollary 12. For any function k and any $\sigma = 4^p, p > 0$ by Theorem 11 we can build an infinite family of strings $\{S_n\}_{n=1}^{\infty}$ such that output generated by any natural parsing method on $S \in \{S_n\}_{n=1}^{\infty}$ is lower bounded by:

$$\mathsf{A}(S) \geq |S|H_k(S) + \frac{|S|k(\log\sigma - \mathcal{O}(1))}{2(2\log_\sigma|S| - k)} \quad . \tag{7}$$

As $k \leq \log_\sigma|S| - \frac{1}{2}$ and the inequality holds for any $\sigma = 4^p$, we can choose big enough σ so that the second summand is $\Omega(\frac{|S|k\log\sigma}{\log_\sigma|S|})$. ◀

Proof of Corollary 13. We use the same construction as in the case of Corollary 12, i.e. we build the string for parameter k. By Lemma 7, the mean of the first j entropies is at least:

$$\frac{1}{j}\left(k\log_\sigma|S| + (j-k)\frac{\log_\sigma|S|}{2} - \mathcal{O}\left(\frac{j\log|S|}{|S|}\right)\right) \geq H_k(S) + \frac{k\log\sigma}{2j} - \mathcal{O}\left(\frac{j\log|S|}{|S|}\right) \ . \tag{8}$$

On the other hand, by (7) we get that the output is lower bounded by (as $2\log_\sigma|S| - k = j$):

$$\mathsf{A}(S) \geq |S|H_k(S) + \frac{|S|k(\log\sigma - \mathcal{O}(1))}{2j} \ . \tag{9}$$

As we can choose $\sigma = 4^p$ arbitrarily we have that for each ϵ there exists such σ and s_0 so that for all constructed strings with $|S| \geq s_0$ the difference (8) will be larger than (9) multiplied by $(1-\epsilon)$. ◀

Generalization of strings from Lemma 8, Proof of Corollary 14

Sketch of the Proof of Lemma 15. We comment on how to modify the construction and proof of Lemma 8; observe that, in essence we replace a constant 2 in Lemma 8 with $r \geq 2$. As a first step, Lemma 8 takes de Bruijn string B of order $2k+1$ over binary alphabet and parse it into phrases of length two in two ways and combine those two strings together. We proceed similarly, but we take de Bruijn string B of order $rk+1$ and instead of parsing B into phrases of length two, we parse it into phrases of length r in r ways and combine all of the r strings. For example for $r=3$ we have:

$$\begin{aligned}
Y_B^1 &= |a_1a_2a_3|a_4a_5a_6|\cdots|a_{n-5}a_{n-4}a_{n-3}|a_{n-2}a_{n-1}a_n| \\
Y_B^2 &= |a_2a_3a_4|a_5a_6a_7|\cdots|a_{n-4}a_{n-3}a_{n-2}|a_{n-1}a_na_1| \\
Y_B^3 &= |a_3a_4a_5|a_6a_7a_8|\cdots|a_{n-3}a_{n-2}a_{n-1}|a_na_1a_2|
\end{aligned}$$

As in proof of Lemma 8, we can choose a shift of B such that $a_1 = a_2 = \cdots = a_5$, so that all those strings begin with the same triple, so after the merging of letters: with the same letter. In this way, the (short enough) cyclic occurrences in the concatenation are the same as cyclic occurrences in the separate strings. Moreover we can use the same argument with constructing the graph so we get the Lemma for arbitrary l.

Concerning the degree of the graph, recall that in the the proof of Lemma 8 each k-letter string over the alphabet $(4^p) = \sigma$ had $(2^p) = \sqrt{\sigma}$ occurrences, all followed by different letters, thus the degree of the graph G_0 (and so each G_i) was $\sqrt{\sigma}$. Now, since we merge r letters into one and the B was a $rk+1$ de Bruijn string, each k-letter string over the alphabet $(2^{rp}) = \sigma$ has $(2^p) = \sqrt[r]{\sigma}$ occurrences, thus the degree of the graph is $\sqrt[r]{\sigma}$. In essence this justifies the replacement of $1/2$ by $1/r$ in all the exponents of σ. Thus we get that that $|S|H_k^{\circlearrowleft}(S) = \frac{1}{r}|S|H_0^{\circlearrowleft}(S)$, and from the Lemma 7 the same (roughly) holds between $|S|H_k(S)$ and $|S|H_0(S)$. Thus we get the generalized version of Theorem 6, which, intuitively, says that we can construct strings whose mean of entropies can be arbitrarily large with respect to $H_k(S)$. ◀

Sketch of the Proof of Lemma 16. We skip the calculations, as they are almost the same as in the proof of Lemma 10, the only difference is that we use different bounds for $|S|_w^{\circlearrowleft}$ (and so for l_w), i.e. for w such that $|S|_w^{\circlearrowleft} > 0$ we have (by Lemma 15):

$$|S|_w^{\circlearrowleft} = \begin{cases} \frac{n}{\sigma^{|w|}} & \text{for } |w| \leq k \\ \frac{n}{\sigma^{|w|/r+(r-1)k/r}} & \text{for } k < |w| \leq z \end{cases} \ . \qquad ◀$$

Proof of Theorem 17. The proof is similar to the one of Theorem 6: we take the string from Lemma 16 and apply Lemma 7 to change from cyclic entropy to entropy. ◀

Proof of Corollary 14. We show that Lemma 16 implies Corollary 14. We construct the family of strings as following: fix p and r and let $\sigma = (2^r)^p$, as in Lemma 15. Take l such that $k = \frac{\alpha}{(1-\alpha)r}(l+1)$ is a natural number. Construct the string S from Lemma 15, its length is $\sigma^{k+\frac{l+1}{r}}$. As r and α are fixed and k is an increasing function of l, for sufficiently large l we have $\alpha \log_\sigma |S| \leq \log_\sigma |S| - \frac{1}{r}$, as this is equivalent to $\frac{1}{r} \leq (1-\alpha)(k + \frac{l+1}{r})$. This means we can construct the family of strings from Lemma 15 for parameter k. By easy calculation

$$\begin{aligned}\alpha \log_\sigma |S| &= \alpha \left(k + \frac{l+1}{r}\right)\\ &= \alpha \left(\frac{\alpha}{1-\alpha} \cdot \frac{l+1}{r} + \frac{l+1}{r}\right)\\ &= \alpha \cdot \frac{\alpha + (1-\alpha)}{1-\alpha} \cdot \frac{l+1}{r}\\ &= \frac{\alpha}{1-\alpha} \cdot \frac{l+1}{r}\\ &= k \enspace ,\end{aligned}$$

as desired. Thus $\frac{k}{\alpha} = \log_\sigma |S| = k + (l+1)/r$. Define $z = k + l + 1$ as in Lemma 16; again by easy calculations $z = k + kr/\alpha - kr = kr/\alpha - k(r-1)$.

To prove that A cannot perform better than $\frac{1}{1-\alpha}|S|H_k(S) + o(|S|\log\sigma)$ we lower bound the ratio of A's output, i.e. $|Y_S|H_0(Y_S)$, and $|S|H_k(S)$ by $\frac{1}{1-\alpha}$. We estimate the former by Lemma 16 and use the same estimation on $|Y_S|\log\frac{|S|}{|Y_S|}$ as in the proof of Theorem 11, i.e. $|Y_S|\log\frac{|S|}{|Y_S|} \leq \lambda|S|$, $\lambda = 0.54$:

$$\begin{aligned}|Y_S|H_0(Y_S) &\geq \frac{|S|(z+(r-1)k)}{r \cdot z}\log\sigma - |Y_S|\log\frac{|S|}{|Y_S|}\\ &\geq \frac{|S|(z+(r-1)k)}{r \cdot z}\log\sigma - \lambda|S| \enspace . \end{aligned} \tag{10}$$

On the other hand, by Theorem 17 we have that:

$$|S|H_k(S) \leq \frac{\log_\sigma |S|}{r} \enspace . \tag{11}$$

Combining these two we obtain:

$$\begin{aligned}\frac{|Y_S|H_0(Y_S)}{|S|H_k(S)} &\geq \frac{|Y_S|H_0(Y_S)}{(|S|\log\sigma)/r} && \text{by (11)}\\ &\geq \frac{z+(r-1)k}{z} - \frac{\lambda|S|r}{|S|\log\sigma} && \text{by (10)}\\ &\geq \frac{kr/\alpha}{kr/\alpha - k(r-1)} - \frac{\lambda|S|r}{|S|rp} && \text{as } z = kr/\alpha - k(r-1),\ \log\sigma = rp\\ &= \frac{r/\alpha}{r/\alpha - (r-1)} - \lambda/p\\ &= \frac{1}{1 - \alpha\cdot\frac{(r-1)}{r}} - \lambda/p \enspace . \end{aligned} \tag{12}$$

Now, if A achieves $\beta|S|H_k(S)+f(|S|\log\sigma)$ bits, for some $f(m)\in o(m)$, and as $|Y_S|H_0(Y_S)$ lower bounds the output size for natural parsers, we have:

$$\begin{aligned}\frac{|Y_S|H_0(Y_S)}{|S|H_k(S)} &\le \beta+\frac{f(|S|\log\sigma)}{|S|H_k(S)}\\ &\le \beta+\frac{f(|S|\log\sigma)}{\frac{|S|\log\sigma}{r}-\mathcal{O}(k\log|S|)} && \text{by Theorem 17}\\ &= \beta+\frac{rf(|S|\log\sigma)}{|S|\log\sigma-\mathcal{O}(rk\log|S|)} \qquad (13)\end{aligned}$$

Combining (12) and (13) yields a lower bound on β:

$$\beta \ge \frac{1}{1-\alpha\cdot\frac{(r-1)}{r}}-\lambda/p-\frac{rf(|S|\log\sigma)}{|S|\log\sigma-\mathcal{O}(rk\log|S|)} \qquad (14)$$

Now, fix r and set $p=r$ in (14), this in particular makes σ fixed as well. Consider the last term in (14), i.e. $\frac{rf(|S|\log\sigma)}{|S|\log\sigma-\mathcal{O}(rk\log|S|)}$. When $|S|\to\infty$, we have $rk\log|S|=o(|S|\log\sigma)$ and so $f(|S|\log\sigma)/|S|\log\sigma$ is arbitrarily small. Thus this term vanishes when $|S|\to\infty$ and so β has to be at least

$$\beta \ge \frac{1}{1-\alpha\cdot\frac{(r-1)}{r}}-\lambda/r\ .$$

This holds for any r, so also for the limit with $r\to\infty$, and so

$$\beta \ge \frac{1}{1-\alpha}\ ,$$

as claimed. ◀