HOLZ: High-Order Entropy Encoding of Lempel–Ziv Factor Distances

Dominik Köppl*, Gonzalo Navarro†, and Nicola Prezza‡

* M&D Data Center
TMDU
Tokyo, Japan
koeppl.dsc@tmd.ac.jp

†Dept. of Computer Science
University of Chile
Santiago, Chile
gnavarro@dcc.uchile.cl

‡DAIS
Ca' Foscari University
Venice, Italy
nicola.prezza@unive.it

Abstract

We propose a new representation of the offsets of the Lempel–Ziv (LZ) factorization based on the co-lexicographic order of the text's prefixes. The selected offsets tend to approach the k-th order empirical entropy. Our evaluations show that this choice is superior to the rightmost and bit-optimal LZ parsings on datasets with small high-order entropy.

1 Introduction

The Lempel–Ziv (LZ) factorization [1] is one of the most popular methods for lossless data compression. It is a *factorization*, that is, splitting the text T into *factors*, each being the longest string that appears before in T, and replacing each factor by a reference to its preceding occurrence (called its *source*).

Most popular compression schemes such as `zip` or `gzip` use a variant called LZ77 [2], which finds sources only within a sliding window w. Though this restriction simplifies compression and encoding, it misses repetitions with gaps larger than $|w|$, and thus compressing k copies of a sufficiently long text T results in a compressed file being about k times larger than the compressed file of T. Such long-spaced repetitions are common in highly-repetitive datasets like genomic collections of sequences from the same taxonomy group, or from documents managed in a revision control system. Highly-repetitive datasets are among the fastest-growing ones in recent decades, and LZ compression is one of the most effective tools to compress them [3]. This is one of the main reasons why the original LZ factorization (i.e., without a window) moved into the spotlight of recent research.

Although LZ catches even distant repetitions, the actual encoding of the factorization is an issue. Each factor is usually represented by its length and the distance to its source (called its *offset*). While the lengths usually exhibit a geometric distribution favorable for universal encoders, the offsets tend to approach a uniform distribution and their codes are long. Since the sources are not uniquely defined, different tie breaks have been exploited in order to improve compression, notably the *rightmost* parsing (i.e., choosing the closest source) and the *bit-optimal* parsing [4] (i.e., optimizing the size of the encoded file instead of the number of factors).

All previous LZ encodings have in common that they encode the *text distance* to the source. In this paper we propose a variant that encodes the distances between

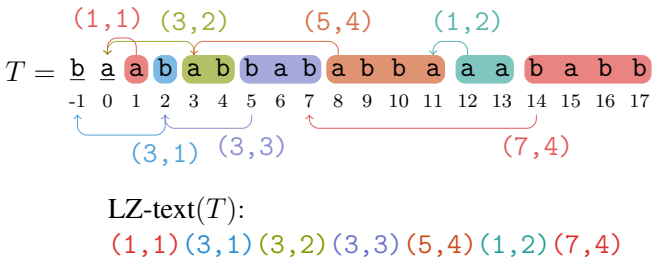

Figure 1: LZ factorization of $T = \texttt{ababbababbaaababb}$ and its LZ-text(T) coding into pairs.

the co-lexicographically sorted prefixes of the processed text, and argue that this choice for the offsets approaches the k-th order empirical entropy of the LZ factors. Our experiments show that this encoding is competitive even with the bit-optimal parsing [4], performing particularly well on texts whose high-order entropy is low.

2 LZ Factorization

Let $T[1..n] \in \Sigma^n$ be a text of length n whose characters are drawn from an integer alphabet $\Sigma = [0..\sigma - 1]$ with $\sigma = n^{\mathcal{O}(1)}$. The LZ factorization is a partitioning of the text T into factors $F_1 \cdots F_z$ such that every factor F_x is equal to the longest substring that starts before F_x in the text (its source). Here, we imagine the text being prefixed by the characters of Σ in non-positive positions, i.e. $T[-c] = c$ for each $c \in \Sigma$. Further, we compute the factorization on $T[1..n]$ while taking the prefixes of all suffixes starting in $T[-\sigma..p-1]$ under consideration for previous occurrences of a factor starting at $T[p]$. In this setting, we always find a factor of length at least one. Hence, each factor F_x can be encoded as a pair $(\mathsf{off}_x^{\mathsf{T}}, \ell_x)$, where $\ell_x = |F_x|$ is the length of F_x, and $\mathsf{off}_x^{\mathsf{T}}$ is the *textual offset* of F_x, that is, the distance of the starting positions of F_x and its source. Thus, the LZ factorization can be represented as a sequence of these integer pairs of offsets and lengths, which we denote by LZ-text(T). This representation is not unique with respect to the choice of the offsets in case of a factor having multiple occurrences that start before it. The so-called *rightmost* parsing selects always the smallest possible offset as a tie break.

Example. Let $T = \texttt{ababbababbaaababb}$. Then the LZ factorization and its representation as pairs are given in Fig. 1. Remember that we imagine the text as being prefixed by all characters of Σ, being $\texttt{ba}$ in this case; we only factorize $T[1..|T|]$, that is, the non-underlined characters of $\underline{\texttt{ba}}\texttt{ababbababbaaababb}$. □

With the rightmost parsing it holds that, for a string drawn from a stationary ergodic source, the minimum number of bits required to write the number $\mathsf{off}_x^{\mathsf{T}}$, which is $\lceil \log_2(\mathsf{off}_x^{\mathsf{T}} + 1) \rceil$, approaches the zero-order entropy of F_x in T (see [5] or [6, Sec. 13.5.1]), with a probability approaching 1 as x grows. Intuitively, this happens because, being $\Pr(F_x = S)$ the probability that the next LZ factor F_x equals the substring S, on expectation we need to visit $\mathsf{off}_x^{\mathsf{T}} = 1/\Pr(F_x = S)$ positions back

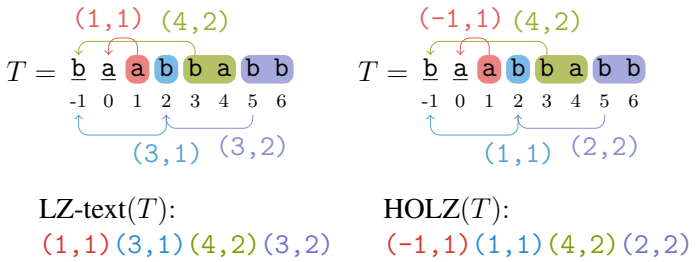

Figure 2: LZ factorization of $T = \texttt{abbabb}$ with its LZ-text (*left*) and HOLZ encoding (*right*). Since the encodings only differ in the values of their offsets, they are equal in length.

(the log of which is F_x's entropy) before encountering an occurrence of S. The fact that the factors are long ($\Theta(\log_\sigma n)$ characters on average [1, Thm. 2]) can be used to show that the size of LZ-text(T) approaches the k-th order entropy of T, for small enough $k = o(\log_\sigma n)$. In that light, we aim to reach the high-order entropy of the factors F_x, by means of changing the encoding of the sources.

3 The HOLZ Encoding

We now give a different representation of the sources of the LZ factors, and argue that this representation achieves high-order entropy. Broadly speaking, we compute the offsets in co-lexicographic order[1] of the text's prefixes, rather than in text order. We denote this new offset as off_p, to distinguish it from the textual offset $\mathsf{off}^{\mathsf{T}}_p$ used in LZ-text(T).

Definition 1. *Let $T_i := T[-\sigma+1..i]$, for $i \in [-\sigma..n]$, be the prefixes of T, including the empty prefix $T_{-\sigma} = \epsilon$. Assume we have computed the factorization of $T[1..p-1]$ for a text position $p \in [1..n]$ and that a new factor F_x of length ℓ_x starts at $T[p]$.*

Let $T_{j_1} \prec T_{j_2} \prec \cdots \prec T_{j_{p+\sigma}}$ be the prefixes $T_{-\sigma}, \ldots, T_{p-1}$ sorted in the co-lexicographic order $\prec$. Let r_{p-1} be the position of T_{p-1} in that order, that is, $j_{r_{p-1}} = p-1$, and let t_{p-1} be the position closest to r_{p-1} (i.e., minimizing $|r_{p-1} - t_{p-1}|$) satisfying $T[j_{t_{p-1}} + 1..j_{t_{p-1}} + \ell_x] = F_x$. Then we define $\mathsf{off}_x := r_{p-1} - t_{p-1}$; note off_x can be negative. We call HOLZ(T) *the resulting encoding of pairs (off_x, ℓ_x), where HO stands for high-order entropy.*

Example. We present a complete step-by-step factorization of the small sample string $T = \texttt{abbabb}$. The HOLZ(T) factorization is given in Fig. 2, and Table 1 depicts the four factorization steps; remember that we do not factorize the added prefix $T[-\sigma+1..0]$. A detailed walk-through follows:

1. For the first factor $F_1 = \texttt{a}$ starting at $p = 1$, the text prefix starting with F_x and being the closest to the r_0-th co-lexicographically smallest prefix, has rank $t_0 = 3$, thus F_1's length and offset are $\ell_1 = 1$ and $\mathsf{off}_1 = r_0 - t_0 = 2 - 3 = -1$, respectively. We then represent $F_1 = T[1] = \texttt{a}$ by the pair $(-1, 1)$.

[1] That is, the lexicographic order of the reversed strings.

Table 1: Step-by-step computation of $\mathbf{HOLZ}(T)$ with $T = \texttt{abbabb} = F_1F_2F_3F_4$. Underlined characters represent the virtual text prefix containing all alphabet's characters.

		sorted prefixes	text to their right	
$T[-1..-2]$	=	ϵ	baabbabb	1
$T[-1..0]$	=	ba	abbabb	$2 = r_0$
$T[-1..-1]$	=	b	aabbabb	$3 = t_0$

Computing $F_1 = T[1]$.

		sorted prefixes	text to their right	
$T[-1..-2]$	=	ϵ	baabbabb	$1 = t_1$
$T[-1..1]$	=	baa	bbabb	$2 = r_1$
$T[-1..0]$	=	ba	abbabb	3
$T[-1..-1]$	=	b	aabbabb	4

Computing $F_2 = T[2]$.

		sorted prefixes	text to their right	
$T[-1..-2]$	=	ϵ	baabbabb	$1 = t_2$
$T[-1..1]$	=	baa	bbabb	2
$T[-1..0]$	=	ba	abbabb	3
$T[-1..-1]$	=	b	aabbabb	4
$T[-1..2]$	=	baab	babb	$5 = r_2$

Computing $F_3 = T[3..4]$.

		sorted prefixes	text to their right	
$T[-1..-2]$	=	ϵ	baabbabb	1
$T[-1..1]$	=	baa	bbabb	$2 = t_4$
$T[-1..0]$	=	ba	abbabb	3
$T[-1..4]$	=	baabba	bb	$4 = r_4$
$T[-1..-1]$	=	b	aabbabb	5
$T[-1..2]$	=	baab	babb	6
$T[-1..3]$	=	baabb	abb	7

Computing $F_4 = T[5..6]$.

2. Next, we update the table of the sorted prefixes, obtaining the order shown on the top-right table. The next factor, starting at position $p = 2$, is $F_2 = \texttt{b}$, so $\ell_2 = 1$ and $\mathsf{off}_2 = r_1 - t_1 = 2 - 1 = 1$. The second pair is thus $(1, 1)$.

3. We update the table of the sorted prefixes, obtaining the order shown on the bottom-left table. This time, $p = 3$, $F_3 = \texttt{ba}$, $\ell_3 = 2$, and $\mathsf{off}_3 = r_2 - t_2 = 5 - 1 = 4$. The third pair is thus $(4, 2)$.

4. We update the table of the sorted prefixes, obtaining the order shown on the bottom-right table; the final pair is $(2, 2)$.

Thus, we obtained $\mathbf{HOLZ}(T) = \texttt{(-1,1) (1,1) (4,2) (2,2)}$. □

Towards High-Order Entropy

Only for the purpose of formalizing this idea, let us define a variant of HOLZ, $\mathbf{HOLZ}^k(T)$, which precedes T with a (virtual) de Bruijn sequence of order $k + 1$, so that every string of length $k + 1$ appears in $T[-\sigma^{k+1} - k + 2..0]$ (i.e., classical $\mathbf{HOLZ}(T)$ is $\mathbf{HOLZ}^0(T)$). We modify the LZ factorization so that F_x, starting at $T[p]$ and preceded by the string S_x of length k, will be the longest prefix of $T[p..]$ such that $S_x \cdot F_x$ appears in T starting before position $p - k$. The resulting factorization $T = F_1F_2\ldots$ has more factors than the LZ factorization, but in exchange, the offsets of the factors F_x are encoded within their k-th order (empirical) entropy. Let $\#S$ be the frequency of substring S in T. Assuming that the occurrences of S_xF_x distribute uniformly among the occurrences of S_x in every prefix of T, the distance $|\mathsf{off}_x|$ between two consecutive sorted prefixes of T suffixed by S_x and followed by F_x is in expectation $\mathbb{E}(|\mathsf{off}_x|) \leq \#S_x/\#S_xF_x$. Then, $\mathbb{E}(\log_2 |\mathsf{off}_x|) \leq \log_2 \mathbb{E}(|\mathsf{off}_x|) \leq \log_2(\#S_x/\#S_xF_x)$ and the total expected size of the encoded offsets is $\mathbb{E}\left(\sum_x \log_2 |\mathsf{off}_x|\right) = \sum_x \mathbb{E}(\log_2 |\mathsf{off}_x|) \leq \sum_x \log_2 \frac{\#S_x}{\#S_xF_x}$.

In the empirical-entropy sense (i.e., interpreting probabilities as relative frequencies in T), the definition of high-order entropy we can reach is restricted to the

factors we produce. Interpreting conditional probability as following in the text, this is $H_k = \sum_x \log_2 \frac{1}{\Pr(F_x|S_x)} = \sum_x \log_2 \frac{\Pr(S_x)}{\Pr(S_x F_x)} = \sum_x \log_2 \frac{\#S_x}{\#S_x F_x}$, that is, the expected length of our encoding is bounded by the k-th order empirical entropy of the factors. This is also the k-th order empirical entropy of the text if we assume that the factors start at random text positions.

Recall that, the longer k, the shorter the phrases, so there is an optimum for a likely small value of k. While this optimum may not be reached by HOLZ (which always chooses the longest phrase), it is reached by the bit-optimal variant of HOLZ that we describe in the next section, *simultaneously* for every k.

Our experimental results validate our synthetic analysis, in the sense that $\mathbf{HOLZ}(T)$ performs better than $\mathbf{LZ\text{-}text}(T)$ on texts with lower k-th order entropy, for small k.

Algorithmic Aspects

For space reasons, here we give only a high-level idea of our algorithm computing $\mathbf{HOLZ}(T)$. The reader can find a more detailed description in the full version of the paper [7]. Our idea follows the framework described in [8]: we build online the Burrows-Wheeler transform [9] of the reversed text, by updating it with the text's characters from the first to last. For this, we use the dynamic compressed string data structure of Munro and Nekrich [10]. At the same time, we search in the BWT the current LZ77 factor F_x. This yields the BWT interval of all previous occurrences of the factor in the text. One step before the interval becomes empty, we pick the occurrence of F_x being the closest (in co-lexicographic order) to the current text prefix and compute the co-lexicographic rank of the text prefix preceding the factor (with $|F_x|$ LF mapping steps on the BWT). Finally, a bitvector marking all BWT characters contained in the new factor allows us skipping those characters while computing the factor's offset. As a result, we can compute **HOLZ** in $n(1 + H_k) + o(n \log \sigma)$ bits of space, for any $k \in o(\log_\sigma n)$, and $\mathcal{O}(n \log n / \log\log n)$ time.

4 The Bit-Optimal HOLZ

Ferragina et al. [4] studied a generalization of the Lempel–Ziv parsing in the sense that they considered for each text position all possible factor candidates (not just the longest ones), optimizing for the representation minimizing a fixed encoding of the integers (e.g. Elias-δ). In other words, in their setting we are free to choose both the offset and factor lengths, thus effectively choosing among all possible unidirectional macro-schemes [11]. Within their framework, LZ can be understood as a greedy factorization that locally always chooses the longest factor among all candidates. This factorization is optimal with respect to the number of computed factors, but not when measuring the *bit-size* of the factors compressed by the chosen encoding for the integers, in general. Given a universal code **enc** for the integers, a *bit-optimal* parsing has the least number of bits among all unidirectional parsings using **enc** to encode their pairs of lengths and offsets. In the setting of textual offsets, [4] proposed an algorithm computing the bit-optimal LZ factorization in $\mathcal{O}(n \lg n)$ time with $\mathcal{O}(n)$ words of space, provided that the code **enc** transforms an integer in the range $[1..n]$

to a bit string of length $\mathcal{O}(\lg n)$. In the following, we take this restriction of enc as granted, as it reflects common encoders like Elias-δ.

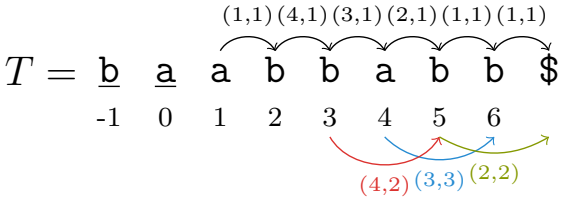

Figure 3: Graph of the factor candidates for LZ-text(T). Every path from node 1 to node $n+1$ gives us a sequence of pairs that can be used alternatively to LZ-text(T), which is obtained by always taking the locally longest edge. Although it is guaranteed that the path for LZ-text(T) has the least number of edges, the compressed representation of the edge labels does not lead to the best compression in general. Using Elias-γ as our encoder enc with $|\mathsf{enc}(x)| = 1$ for $x = 1$, $|\mathsf{enc}(x)| = 2$ for $x \in \{2, 3\}$, and $|\mathsf{enc}(x)| = 3$ for $x \in [4..7]$, the compressed size of LZ-text(T) taking the red and the green edge is $1+1+3+1+3+2+2+2 = 15$ bits. If we exchange the red and green edge with one blue edge and two black edges, we obtain $1+1+3+1+1+1+2+2+1+1 = 14$ bits.

The idea of Ferragina et al. [4] is to build a *factor graph* (cf. Fig. 3) representing all possible choices of factors: the nodes are the text positions, and an edge between two positions indicates that the substring between those two positions has a previous occurrence. The weight of the edge is the number of bits used by the chosen encoder to encode the distance between the factor and its source. The bit-optimal LZ77 factorization corresponds to the minimum-weight path connecting the first and last text position. Even though the number of edges of this graph could be quadratic, Ferragina et al. [4] observed that edges leaving the same text position and having equal weight are redundant, so we can keep the one spanning the largest number of text positions. This edge is called *maximal*. Since there are at most $O(\log n)$ distinct weights, this observation allows to reduce the number of edges to just $O(n \log n)$.

Our Algorithm

For space reasons, here we give only a high-level idea of how the strategy of Ferragina et al. [4] can be generalized to the HOLZ encoding. The reader can find a more detailed description in the full version of the paper [7]. We define a *factor graph* similarly to Ferragina et al., with the difference that, in our case, edge weights correspond to the number of bits required by the chosen encoder to represent offsets in the co-lexicographic order of the text's prefixes. The major problem is finding the maximal edges efficiently. We solve this problem by maintaining a dynamic wavelet tree [12] (a two-dimensional grid) mapping the co-lexicographic ranks of the processed text's prefixes $T[-\sigma..i]$ (that is, inverse suffix array values of the reversed text) to the lexicographic ranks of the corresponding text's suffixes $T[i+1..n]$ (that is, inverse suffix array values of the text). These two-dimensional points are inserted in the grid as the text is processed left-to-right. We also compute the longest-common prefix (LCP) array and a range-minimum query (RMQ) data structure on the LCP array. These two data structures are computed on the whole text in a preprocessing step. Using these data structures, for any range $[a..b]$ centered on the co-lexicographic rank

Table 2: 20 MB prefixes of the Pizza&Chili corpus datasets. H_k denotes the k-th order empirical entropy. z is the length of LZ-text(T), and r is the number of equal-letter runs in the BWT.

dataset	σ	z	r	H_0	H_1	H_2	H_3	H_4
CERE	5	8492391	1060062	2.20	1.79	1.79	1.78	1.78
COREUTILS	235	3010281	910043	5.45	4.09	2.84	1.85	1.31
DBLP.XML	96	3042484	834349	5.22	3.26	1.94	1.26	0.89
DNA	14	12706704	1567099	1.98	1.93	1.92	1.92	1.91
E.COLI	11	8834711	1146785	1.99	1.98	1.96	1.95	1.94
ENGLISH	143	5478169	1277729	4.53	3.58	2.89	2.33	1.94
INFLUENZA	15	876677	210728	1.97	1.93	1.93	1.92	1.91
KERNEL	160	1667038	488869	5.38	4.00	2.87	1.98	1.47
PARA	5	8254129	1028222	2.17	1.83	1.83	1.82	1.82
PITCHES	129	10407645	2816494	5.62	4.85	4.28	3.50	2.18
PROTEINS	25	8499596	1958634	4.20	4.17	4.07	3.71	2.97
SOURCES	111	4878823	1361892	5.52	4.06	2.98	2.13	1.60
WORLDLEADERS	89	408308	129146	4.09	2.46	1.74	1.16	0.73

j of a given text prefix, one can find the source (in $[a..b]$) maximizing the factor's length. An exponential search on $[a..b]$ – repeated for each $1 \leq j \leq n + \sigma$ — yields maximal edges for each possible text position and edge weight.

An analysis of this algorithm reveals that the running time is $\mathcal{O}(n \lg^3 n)$ and the working space is $\mathcal{O}(n \lg n)$ words. The space could be improved to $\mathcal{O}(n)$ words using the same techniques discussed in [4]. Decompression works in both variants (HOLZ or its bit-optimal variant) in the same way. The idea is to use a dynamic BWT of the current reversed text prefix, and use it to extract the next factor. Overall, using the dynamic string data structure of [10], the decompression algorithm runs in $O(n \log n / \log \log n)$ time and uses $nH_k + o(n \log \sigma)$ bits of space (excluding the input, which however can be streamed).

5 Experiments

For our experiments, we focused on the Canterbury (`corpus.canterbury.ac.nz`) and Pizza&Chili (`pizzachili.dcc.uchile.cl`) datasets. For the latter, we only processed the first 20 MB of each file. See Tables 2 and 3 for some characteristics of these datasets. The datasets KENNEDY.XLS, PTT5, and SUM contain '0' bytes, which is a prohibited value for some used tools like suffix array construction algorithms. In a precomputation step for these files, we escaped each '0' byte with the byte pair '254' '1' and each former occurrence of '254' with the pair '254' '254'.

For comparison, we used LZ-text(T) and the bit-optimal implementation of [13], referred to as bitopt in the following. For the former, we remember that the choice of the offsets can be ambiguous. Here, we select two different codings that break ties in a systematic manner: The rightmost parsing discussed in the introduction, and the output of an algorithm [14, 15] computing LZ with next-smaller value (NSV) and previous-smaller value (PSV) arrays built on the suffix array. Although the compressed output is at least that of the rightmost parsing, this algorithm runs in

Table 3: Datasets from the Canterbury corpus; n is the number of text characters. See Table 2 for a description of the other columns.

dataset	n	σ	z	r	H_0	H_1	H_2	H_3	H_4
ALICE29.TXT	152089	74	66903	22897	4.56	3.41	2.48	1.77	1.32
ASYOULIK.TXT	125179	68	62366	21634	4.80	3.41	2.53	1.89	1.37
CP.HTML	24603	86	9199	4577	5.22	3.46	1.73	0.77	0.44
FIELDS.C	11150	90	3411	1868	5.00	2.95	1.47	0.86	0.62
GRAMMAR.LSP	3721	76	1345	853	4.63	2.80	1.28	0.67	0.44
KENNEDY.XLS	1486290	255	219649	145097	3.13	2.04	1.76	1.19	1.12
LCET10.TXT	426754	84	165711	52594	4.66	3.49	2.61	1.83	1.37
PLRABN12.TXT	481861	81	243559	72622	4.53	3.36	2.71	2.13	1.72
PTT5	961861	158	65867	25331	1.60	0.47	0.39	0.31	0.26
SUM	50503	254	13544	7826	4.76	2.52	1.61	1.15	0.90
XARGS.1	4227	74	2010	1172	4.90	3.19	1.55	0.72	0.42

linear time, whereas we are unaware of an algorithm computing the rightmost parsing in linear time – the currently best algorithm needs $\mathcal{O}(n + n \log \sigma / \sqrt{\log n})$ time [16]. We call these two specializations of LZ-text(T) rightmost and nsvpsv, respectively. We write holz and holz-opt for our original and bit-optimal algorithms computing HOLZ(T), respectively. We selected Elias-δ encoding as the function enc, and present the measured compression ratios in Fig. 4.

We observe that bitopt uses a different variant of the LZ factorization that does not use the imaginary prefix $T[-\sigma..0]$ for references. Instead, it introduces *literal factors* that catch leftmost occurrences characters appearing in T. Their idea is to store a literal factor $S \in \Sigma^+$ by its length $|S|$ encoded in 32-bits, followed by the characters byte-encoded. In the worst case, they pay 40σ bits. Although this overhead is noticeable for small files such as GRAMMAR.LSP — where 40σ bits are roughly 20% of their output size — or CP.HTML — where less than 4% of the output size can be accounted for the literal factors — this fraction ($40\sigma/(8 \cdot 20 \cdot 10^6 \cdot c)$ in Table 2 and $40\sigma/(8n \cdot c)$ in Table 3, where c is the compression ratio shown in Fig. 4) is vanishing for all larger datasets.

Discussion The overall trend that we observe is that our encoding scheme HOLZ performs better than LZ on datasets characterized by a small high-order entropy. More in detail, `holz-bitopt` compresses better than `bitopt` on all 8 datasets having $H_4 \leq 1$, and on 11 over 14 datasets with $H_4 \leq 1.5$. In general, `holz-bitopt` performed no worse (strictly better in all cases except one) than `bitopt` on 14 over 24 datasets. The trend is similar when comparing the versions of the algorithms that always maximize factor length: `holz` and `rightmost`. These results support the intuition that HOLZ is able to exploit high-order entropy, improving — when it is small enough — the compression ratio of the offsets with respect to LZ.

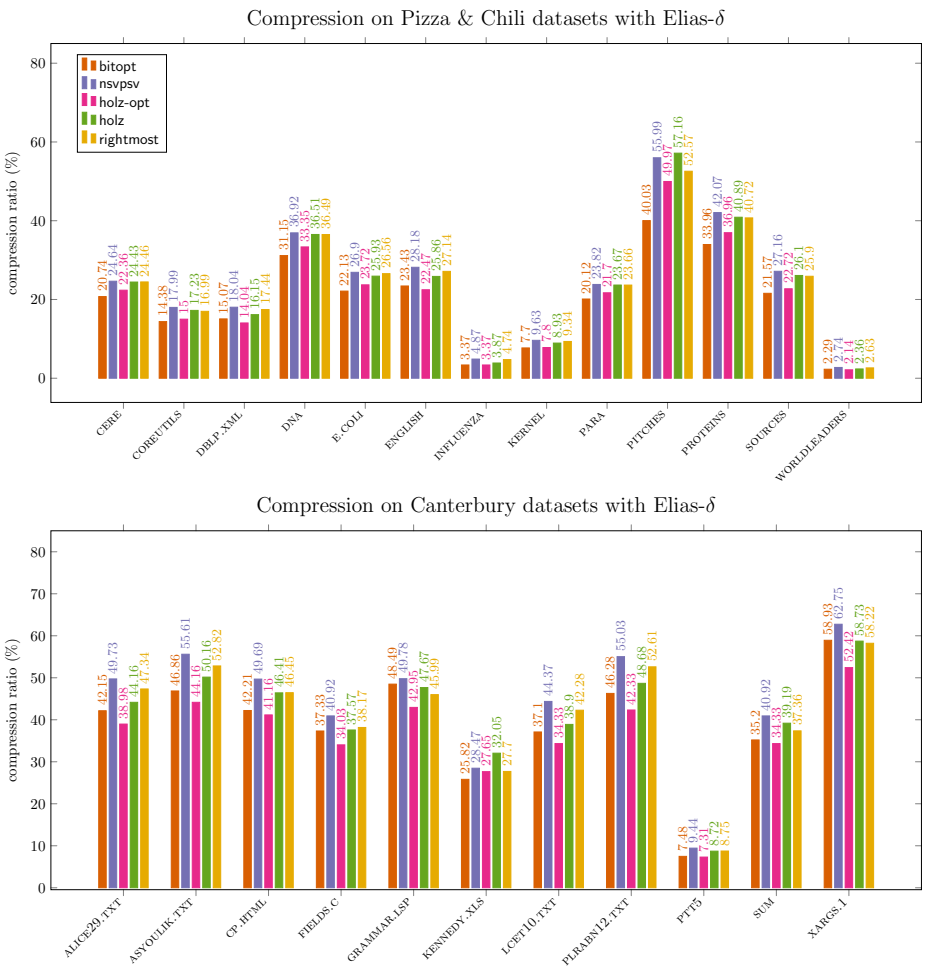

Figure 4: Compression ratios of different encodings and parsings studied in this paper.

6 Open Problems

We wonder whether there is a connection between our proposed encoding **HOLZ** and the Burrows-Wheeler transform (BWT) combined with move-to-front (MTF) coding, which achieves the k-th order entropy of T with $k = \Theta(\log_\sigma n)$. Applying MTF to BWT basically produces a list of pointers, where each pointer refers to the closest previous occurrence of a character whose context is given by the lexicographic order of its succeeding suffix. The difference is that this technique works on characters rather than on factors. Also, we would like to find string families where the compressed output of **HOLZ**(T) is asymptotically smaller than **LZ-text**(T), or vice-versa.

Our current implementation using dynamic wavelet trees is quite slow. Alternatively, for the encoding process we could use a static BWT and a dynamic prefix sum structure to mark visited prefixes in co-lexicographic order, which should be faster than a dynamic BWT. A more promising alternative would be to not use dynamic structures. We are confident that a good heuristic by hashing prefixes according to

some locality-sensitive hash function (sensitive to the co-lexicographic order) will find matches much faster. Note that using the context of length k has the additional benefit to reduce the search space (compared to standard LZ), therefore the algorithm could be much faster and it could be easier to find potential matches.

Acknowledgements This research was funded by JSPS KAKENHI with grant numbers JP21H05847 and JP21K17701, by Basal Funds FB0001 and Fondecyt Grant 1-200038 (Chile), and by Ca' Foscari University under the funding scheme *Ricerca Base - Fondi Primo Insediamento EST (INCINTV)*. We are grateful to github user MitI_7 for some guidance regarding the dynamic wavelet tree implementation.

[1] A. Lempel and J. Ziv, "On the complexity of finite sequences," *IEEE Trans. Inf. Theory*, vol. 22, no. 1, pp. 75–81, 1976.

[2] J. Ziv and A. Lempel, "A universal algorithm for sequential data compression," *IEEE Trans. Inf. Theory*, vol. 23, no. 3, pp. 337–343, 1977.

[3] G. Navarro, "Indexing highly repetitive string collections, part I: Repetitiveness measures," *ACM Comput. Surv.*, vol. 54, no. 2, pp. 29:1–29:31, 2021.

[4] P. Ferragina, I. Nitto, and R. Venturini, "On the bit-complexity of Lempel–Ziv compression," *SIAM J. Comput.*, vol. 42, no. 4, pp. 1521–1541, 2013.

[5] A. D. Wyner and J. Ziv, "The sliding-window Lempel-Ziv algorithm is asymptotically optimal," *Proc. IEEE*, vol. 82, no. 6, pp. 872–877, 1994.

[6] T. M. Cover and J. A. Thomas, *Elements of Information Theory*, Wiley, 2012.

[7] D. Köppl, G. Navarro, and N. Prezza, "HOLZ: high-order entropy encoding of Lempel-Ziv factor distances," 2021, `https://arxiv.org/abs/2111.02478`.

[8] A. Policriti and N. Prezza, "LZ77 computation based on the run-length encoded BWT," *Algorithmica*, vol. 80, no. 7, pp. 1986–2011, 2018.

[9] M. Burrows and D. J. Wheeler, "A block sorting lossless data compression algorithm," Tech. Rep. 124, Digital Equipment Corporation, 1994.

[10] J. Ian Munro and Yakov Nekrich, "Compressed data structures for dynamic sequences," in *Proc. ESA*, 2015, vol. 9294 of *LNCS*, pp. 891–902.

[11] J. A. Storer and T. G. Szymanski, "Data compression via textural substitution," *J. ACM*, vol. 29, no. 4, pp. 928–951, 1982.

[12] R. Grossi, A. Gupta, and J. S. Vitter, "High-order entropy-compressed text indexes," in *Proc. SODA*, 2003, pp. 841–850.

[13] A. Farruggia, P. Ferragina, A. Frangioni, and R. Venturini, "Bicriteria data compression," *SIAM J. Comput.*, vol. 48, no. 5, pp. 1603–1642, 2019.

[14] Maxime Crochemore, Lucian Ilie, and William F. Smyth, "A simple algorithm for computing the lempel ziv factorization," in *Proc. DCC*, 2008, pp. 482–488.

[15] E. Ohlebusch and S. Gog, "Lempel–Ziv factorization revisited," in *Proc. CPM*, 2011, pp. 15–26.

[16] D. Belazzougui and S. J. Puglisi, "Range predecessor and Lempel–Ziv parsing," in *Proc. SODA*, 2016, pp. 2053–2071.