Lempel–Ziv (LZ77) Factorization in Sublinear Time

Dominik Kempa
Department of Computer Science
Stony Brook University
Stony Brook, NY, USA
kempa@cs.stonybrook.edu

Tomasz Kociumaka
Max Planck Institute for Informatics
Saarland Informatics Campus
Saarbrücken, Germany
tomasz.kociumaka@mpi-inf.mpg.de

***Abstract*—Lempel–Ziv (LZ77) factorization is a fundamental problem in string processing: Greedily partition a given string T from left to right into blocks (called *phrases*) so that each phrase is either the leftmost occurrence of a single letter or the longest prefix of the unprocessed suffix that has another occurrence earlier in the text. This simple routine has numerous applications. Most importantly, the LZ77 factorization is the central component and the computational bottleneck of most existing compression algorithms (utilized in formats like `zip`, `pdf`, and `png`). LZ77 is also a widely used algorithmic tool for the detection of repetitions and periodicities in strings, and the centerpiece of many powerful compressed indexes that enable computation directly over compressed data. LZ77 factorization is one of the most studied problems in string processing. In the 47 years since its inception, numerous efficient algorithms were developed for different models of computation, including parallel, GPU, external-memory, and quantum. Remarkably, however, the complexity of the most basic problem is still not settled: All existing algorithms in the RAM model run in $\Omega(n)$ time, which is a $\Theta(\log n)$ factor away from the lower bound of $\Omega(n/\log n)$ (following simply from the necessity to read the entire input, which takes $\Theta(n/\log n)$ space for any $T \in \{0,1\}^n$). Sublinear-time algorithms are known for nearly all other fundamental problems on strings, but LZ77 seems resistant to all currently known techniques.**

We present the first $o(n)$-time algorithm for constructing the LZ77 factorization, breaking the linear-time barrier present for nearly 50 years. More precisely, we show that, in the standard RAM model, it is possible to compute the LZ77 factorization of a given length-n string $T \in \{0,1\}^n$ in $\mathcal{O}(n/\sqrt{\log n}) = o(n)$ time and using the optimal $\mathcal{O}(n/\log n)$ working space. Our algorithm generalizes to larger alphabets $\Sigma = [0\mathinner{.\,.}\sigma)$, where $\sigma = n^{\mathcal{O}(1)}$. The runtime and working space then become $\mathcal{O}((n\log\sigma)/\sqrt{\log n})$ and $\mathcal{O}(n/\log_\sigma n)$, respectively. To achieve this sublinear-time LZ77 algorithm, we prove a more general result: We show that, for any constant $\epsilon \in (0,1)$ and string $T \in [0\mathinner{.\,.}\sigma)^n$, in $\mathcal{O}((n\log\sigma)/\sqrt{\log n})$ time and using $\mathcal{O}(n/\log_\sigma n)$ working space, we can construct an index of optimal size $\mathcal{O}(n/\log_\sigma n)$ that, given any substring $P = T[j\mathinner{.\,.}j+\ell)$ specified with a pair (j,ℓ), computes the leftmost occurrence of P in T in $\mathcal{O}(\log^\epsilon n)$ time. In other words, we solve the indexing/online variant of the LZ77 problem, where we can efficiently query the phrase length starting at any position. Our solution is based on a new type of queries that we call *prefix range minimum queries* or *prefix RMQ*. After developing an efficient solution for these queries, we provide a general reduction showing that any new tradeoff for the prefix RMQ implies a new tradeoff for an index finding leftmost occurrences (and hence a new LZ77 factorization algorithm).

This work has been partially supported by the NSF CAREER Award 2337891 and the Simons Foundation Junior Faculty Fellowship.

A full version of this paper is available at arxiv.org/abs/2409.12146 [1].

***Index Terms*—data compression, Lempel–Ziv compression, Lempel–Ziv factorization, LZ77, prefix range minimum queries**

I. Introduction

The Lempel–Ziv (LZ77) factorization [2], [3] is one of the most fundamental concepts in data compression. In this method, we partition the input string T into a sequence of blocks $T = f_1 f_2 \cdots f_z$. Each block, called a "*phrase*", is either (a) the first occurrence of a letter or (b) a substring that has an earlier occurrence in T. We then encode each phrase f_j either explicitly (in case (a)) or as a pair (ℓ, i), where $\ell = |f_j|$ and i is the position of an earlier occurrence of f_j (in case (b)). Such representation needs $\mathcal{O}(z)$ space and the greedy approach, where T is decomposed from left to right into the longest possible phrases, has been shown to minimize the number of phrases z [2, Theorem 1].

Due to its excellent practical performance, strong theoretical guarantees, and numerous applications, the above algorithm went on to become one of the most widely used compression methods. In 2004, LZ77 was named the IEEE Milestone [4], and in 2021 Jacob Ziv was awarded the IEEE Medal of Honor [5] (the highest IEEE recognition) for his work on LZ77 and its variant LZ78 [6]. Below, we list some of the applications of LZ77.

- LZ77 is the most common compression method: 57 out of 207 compressors in the Large Text Compression Benchmark [7] use it as their main algorithm. It is used in `png`, `pdf`, `zip`, `xz`, `7z`, `gz`, and `arj` formats (to name a few), and in virtually all modern web browsers and web servers [8].
- LZ77 underlies compressed text indexes supporting random access [9]–[13], longest common extension (LCE) queries [13]–[16], rank and select queries [12], [17], [18], pattern matching [19]–[27], and suffix array functionality [28];[1] see [32], [33] for a recent survey.
- LZ77, together with the closely related grammar compression [29], is the framework of algorithms operating directly on compressed data to solve many central problems, including longest common subsequence and

[1]Some of the indexes use the closely related notion of grammars [29] or string attractors [30], instead of building directly on LZ77. However, since computing the smallest grammar and the smaller string attractor is NP-complete [29], [30], the LZ77-based approximations, such as [9], [29], [31], are used in most cases.

edit distance [34]–[36], Hamming distance [37], [38], exact [39]–[41] and approximate pattern matching [42]–[45], and matrix-vector multiplication [46].

- LZ77 is one of the most widely used measures of repetitiveness [32], [33], and it comes with solid mathematical foundations: As shown in [9], [13], [29], [30], [47]–[49], LZ77 is up to logarithmic factors equivalent to grammar compression [29], LZ-End [50], run-length-encoded Burrows–Wheeler Transform [51], macro schemes [52], collage systems [53], string attractors [30], and substring complexity [49]. However, whilst many of the those measures are NP-hard to optimize [29], [30], [52], [53], LZ77 can be constructed in linear time [54]. Moreover, it is one of the smallest measures in practice [33], [50].
- LZ77 factorization is the central tool used for efficient detection of regularities in strings: repetitions [55], runs (maximal repeats) [56]–[59], repeats with a fixed gap [60], approximate repetitions [61], tandem repeats [62], sequence alignments [63], local periods [64], and seeds [65]. These regularities, in turn, have applications in bioinformatics, data mining, and combinatorics; see [66]–[68].

In nearly all applications above, finding the LZ77 factorization is the computational bottleneck. This applies in data compression [8], [69], detection of repetitions [58], [68], as well as index construction: given the LZ77 factorization of the input text, recent algorithms [28], [48] can construct text indexes in *compressed time* (i.e., $\mathcal{O}(z \operatorname{polylog} n)$, where $n = |T|$), which for highly repetitive texts is orders of magnitude smaller than the original (uncompressed) text [33]. Thus, LZ77 factorization is the dominant step.

Algorithms for efficient LZ77 factorization are known in nearly all models of computation, including parallel [70]–[75], GPU [69], [76], [77], external memory [78], [79], and quantum [80] models. LZ77 factorization has also been studied in the dynamic setting [25] and for general (non-integer) alphabets, where the known bounds for factorizing the length-n string with σ distinct characters are $\Theta(n \log \sigma)$ symbol comparisons (for ordered alphabets) [81] or $\Theta(n\sigma)$ symbol equality tests (for unordered alphabets) [82].

In this paper, we focus on the most fundamental and most studied variant, i.e., LZ77 factorization in the static setting in the standard RAM model [83] with the word size $w \geq \log n$. In this model, the input text $T \in [0\,.\,.\,\sigma)^n$ of length n over integer alphabet $\Sigma = [0\,.\,.\,\sigma)$ is represented using $n \log \sigma$ bits, or $\Theta(n/\log_\sigma n)$ machine words.[2] The trivial lower bound for the runtime in this model, following from the necessity to read the input, is $\Omega(n/\log_\sigma n)$. Since the number of LZ77 phrases satisfies $z = \mathcal{O}(n/\log_\sigma n)$ for every text $T \in [0\,.\,.\,\sigma)^n$ [2, Theorem 2], an algorithm running in $\mathcal{O}(n/\log_\sigma n)$ is hypothetically plausible.

The first efficient algorithm for constructing the LZ77 factorization was proposed in 1981 [54]. The algorithm is based on suffix trees [84] and achieves $\mathcal{O}(n \log \sigma)$ time and $\mathcal{O}(n)$ space. The first $\mathcal{O}(n)$-time algorithm (independent of the alphabet size) was given in [59]. Numerous other linear or near linear-time algorithms using $\Theta(n)$ space in the worst case followed [85]–[92], aiming to reduce the runtime or space usage in practice. The first algorithm to reduce the space complexity achieved $\mathcal{O}(n \log^3 n)$ time in the optimal $\mathcal{O}(n/\log_\sigma n)$ space [93]. Subsequent works lowered the time (while keeping the $\mathcal{O}(n/\log_\sigma n)$ space) to $\mathcal{O}(n \log^2 n)$ [94], $\mathcal{O}(n \log n \cdot (1 + \frac{\log \sigma}{(\log\log n)^2}))$ [20], $\mathcal{O}(n \log n)$ [95]–[97], $\mathcal{O}(n(\log \sigma + \log\log n))$ [98], $\mathcal{O}(n \log \sigma)$ [85], $\mathcal{O}(n \log\log \sigma)$ [99], randomized $\mathcal{O}(n)$ [99], and finally, by combining [100] and [101], to deterministic $\mathcal{O}(n)$ time. There also exist algorithms whose runtime depends on z: The procedures in [102], [103] achieve the time complexity of $\mathcal{O}(n/\log_\sigma n + z \operatorname{polylog} n)$.[3] For sufficiently small z, this is $\mathcal{O}(n/\log_\sigma n)$, but these algorithms still require $\Omega(n)$ time in the worst case. Summing up, all prior algorithms to compute the LZ77 factorization need $\Omega(n)$ time in the worst case.

Given the fundamental role of LZ77, we thus ask:

Can we compute the LZ77 factorization of a string $T \in [0\,.\,.\,\sigma)^n$ faster than in $\mathcal{O}(n)$ time?

a) Our Results: After nearly 50 years since the invention of LZ77, we present the first algorithm to compute the LZ77 factorization in $o(n)$ time. For a binary alphabet ($\sigma = 2$), our algorithm runs in $\mathcal{O}(n/\sqrt{\log n})$ time and uses the optimal $\mathcal{O}(n/\log n)$ space. For an integer alphabet $\Sigma = [0\,.\,.\,\sigma)$, it runs in $\mathcal{O}((n \log \sigma)/\sqrt{\log n})$ time and uses $\mathcal{O}(n/\log_\sigma n)$ space. We obtain the same complexities for a variant of LZ77 that prohibits overlaps between phrases and their previous occurrences.[4] (This variant is sometimes preferred in practice since it simplifies the usage of the factorization.) All our algorithms are deterministic.

Theorem I.1 (LZ77 factorization)**.** *Given the $\mathcal{O}(n/\log_\sigma n)$-space representation of a text $T \in [0\,.\,.\,\sigma)^n$, the overlapping and non-overlapping LZ77 factorization of T can be constructed in $\mathcal{O}((n \log \sigma)/\sqrt{\log n})$ time and $\mathcal{O}(n/\log_\sigma n)$ working space.*

We achieve this result as a simple corollary of a much more general tool that we develop. Namely, we propose the first index with sublinear construction that can quickly locate the leftmost occurrences of substrings of T. More precisely, we show that, given any constant $\epsilon \in (0, 1)$ and the $\mathcal{O}(n/\log_\sigma n)$-space representation of $T \in [0\,.\,.\,\sigma)^n$, where $2 \leq \sigma < n^{1/7}$, in $\mathcal{O}((n \log \sigma)/\sqrt{\log n})$ time[5] and $\mathcal{O}(n/\log_\sigma n)$ working space, we can construct an index that, for any position $j \in [1\,.\,.\,n]$ and any length $\ell \in [1\,.\,.\,n + 1 - j]$,[6] in $\mathcal{O}(\log^\epsilon n)$ time returns

[2]Unless indicated otherwise, we measure the space in machine words.

[3]The complexity of the algorithm in [102] is originally stated as $\mathcal{O}(n/\log_\sigma n + z \operatorname{polylog} n + r \operatorname{polylog} n)$, where r is the number of equal-letter runs in the Burrows–Wheeler transform (BWT) [51] of T, but this can be simplified to $\mathcal{O}(n/\log_\sigma n + z \operatorname{polylog} n)$ due to the more recent upper bound $r = \mathcal{O}(z \log^2 n)$ [48].

[4]The number of phrases z_{no} in this variant satisfies $z_{\text{no}} = \mathcal{O}(n/\log_\sigma n)$, as we prove for completeness is the full version [1].

[5]We actually achieve a slightly better time of $\mathcal{O}(n \min(1, \log \sigma/\sqrt{\log n}))$, but for simplicity we use the basic bound. If $\sigma \geq n^{1/7}$, Theorem I.1 follows from standard linear-time solutions [59], [104] because $\log \sigma = \Theta(\log n)$.

[6]For $i, j \in \mathbb{Z}$, denote $[i\,.\,.\,j] = \{k \in \mathbb{Z} : i \leq k \leq j\}$, $[i\,.\,.\,j) = \{k \in \mathbb{Z} : i \leq k < j\}$, and $(i\,.\,.\,j] = \{k \in \mathbb{Z} : i < k \leq j\}$.

the position $\min \mathrm{Occ}(P, T)$ for $P = T[j \mathinner{.\,.} j+\ell)$, where

$$\mathrm{Occ}(P,T) = \\ \{i \in [1 \mathinner{.\,.} n] : i + |P| \le n+1 \text{ and } T[i \mathinner{.\,.} i+|P|) = P\}$$

consists of the starting positions of the occurrences of P in T. Our index also works for *explicit* patterns: given the $\mathcal{O}(m/\log_\sigma n)$-space representation of any pattern $P \in [0 \mathinner{.\,.} \sigma)^m$ satisfying $\mathrm{Occ}(P,T) \neq \emptyset$, we can in $\mathcal{O}(\log^\epsilon n + m/\log_\sigma n)$ time compute $\min \mathrm{Occ}(P,T)$. Observe that, given such data structure, it is easy to compute the overlapping and non-overlapping versions of LZ77 simply by binary searching the length of each phrase. More precisely, computing the length ℓ and the position i of the previous occurrence of a phrase starting at any position j in T takes $\mathcal{O}(\log \ell \cdot \log^\epsilon n)$ time. Across all z phrases of total length n, this sums up to $\mathcal{O}(z \log(n/z) \cdot \log^\epsilon n)$ since logarithm is a concave function. Due to $z = \mathcal{O}(n/\log_\sigma n)$, it thus suffices to set $\epsilon < \frac{1}{2}$ to achieve the worst-case running time of $\mathcal{O}(n/\log_\sigma n \cdot \log\log_\sigma n \cdot \log^\epsilon n) = \mathcal{O}((n\log\sigma)/\sqrt{\log n})$. This method works for both the overlapping and non-overlapping variants of LZ77. However, rather than applying the above strategy directly, we go one step further and generalize the above idea so that, after additional sublinear preprocessing, the computation of the pair (ℓ, i) in the above scenario takes $\mathcal{O}(\log^\epsilon n)$ time (rather than $\mathcal{O}(\log \ell \cdot \log^\epsilon n)$). In other words, we obtain a data structure that provides $\mathcal{O}(\log^\epsilon n)$-time access to the so-called *Longest Previous Factor (LPF)* [105] and the *Longest Previous non-overlapping Factor (LPnF)* [104] arrays. The following result applied with $\epsilon \le \frac{1}{2}$ thus yields Theorem I.1.

Theorem I.2 (LPF and LPnF Index)**.** *Given any constant $\epsilon \in (0,1)$ and the $\mathcal{O}(n/\log_\sigma n)$-space representation of a text $T \in [0 \mathinner{.\,.} \sigma)^n$, where $2 \le \sigma < n^{1/7}$, we can in $\mathcal{O}((n\log\sigma)/\sqrt{\log n})$ time and $\mathcal{O}(n/\log_\sigma n)$ working space construct a data structure of size $\mathcal{O}(n/\log_\sigma n)$ that, given any $j \in [1 \mathinner{.\,.} n]$, in $\mathcal{O}(\log^\epsilon n)$ time returns $\mathrm{LPF}_T[j]$, $\mathrm{LPnF}_T[j]$, $\mathrm{LPFMinOcc}_T[j]$, and $\mathrm{LPnFMinOcc}_T[j]$, which are defined as the length ℓ and the leftmost occurrence of the longest previous (non-overlapping) factor $T[j \mathinner{.\,.} j+\ell)$.*

As discussed above, the central technical result of our paper is a space-efficient index that quickly locates leftmost occurrences of substrings in the text and admits a sublinear-time construction algorithm. The main obstacle to obtaining such an index using prior techniques is that locating leftmost occurrences is typically achieved using *Range Minimum Queries (RMQ)* on top of the suffix array. Although RMQ queries only add $\mathcal{O}(m)$ bits on top of the length-m array they augment [106], their construction needs $\Omega(m)$ time, which prevents achieving $o(n)$-time construction for a length-n text. We instead exploit a sampling-based approach [107], where the idea is to first carefully compute a sample $\mathsf{S} \subseteq [1 \mathinner{.\,.} n]$ of representative text positions within nonperiodic regions of the text (periodic regions are handled separately) such that $|\mathsf{S}| = \mathcal{O}(n/\log_\sigma n)$ [107], and then reduce the queries on the text to orthogonal range queries on a set of points defined by the set S. The natural query corresponding to finding leftmost occurrences is then a 4-sided orthogonal RMQ query. Indeed, a reduction to orthogonal range queries underlies some of the fastest indexes of size $\mathcal{O}(z \operatorname{polylog} n)$ [26]–[28], but it does not lead to an efficient solution in our scenario because no fast construction is known for efficient orthogonal RMQ data structures (such as [108]). We instead propose to replace the general RMQ queries on a plane with a new type of query we call *prefix RMQ*. Given an array of integers $A[1 \mathinner{.\,.} m]$ and sequence S of m strings over alphabet Σ, the prefix RMQ query with arguments $b, e \in [0 \mathinner{.\,.} m]$ and $X \in \Sigma^*$ asks to compute the position that minimizes the value $A[i]$ among all indices $i \in (b \mathinner{.\,.} e]$ for which X is a prefix of $S[i]$. This variant of a 4-sided RMQ query is precisely the specialization that we need to support on S. We propose a space-efficient data structure for prefix RMQ queries and describe its fast construction. Furthermore, by carefully handling periodic regions of the text, where we again prove that the orthogonal RMQ queries have a special structure that supports faster queries, we achieve the following very general reduction from prefix RMQ queries. Plugging our specific tradeoff (discussed in the full version [1]) to this reduction yields our main index. Observe that this reduction is very efficient: aside from an extra $\mathcal{O}(\log\log n)$ term in the query time, prefix RMQ dominate all the complexities.

Theorem I.3 (Index for Leftmost Occurrences)**.** *Consider a data structure answering prefix RMQ queries that, for any sequence of k length-ℓ strings over alphabet $[0 \mathinner{.\,.} \sigma)$, achieves the following complexities:*

1) *Space usage $S(k,\ell,\sigma)$,*
2) *Preprocessing time $P_t(k,\ell,\sigma)$,*
3) *Preprocessing space $P_s(k,\ell,\sigma)$, and*
4) *Query time $Q(k,\ell,\sigma)$.*

For every text $T \in [0 \mathinner{.\,.} \sigma)^n$ with $2 \le \sigma < n^{1/7}$, there exists $k = \mathcal{O}(n/\log_\sigma n)$ and $\ell = \mathcal{O}(\log_\sigma n)$ such that, given the $\mathcal{O}(n/\log_\sigma n)$-space representation of T, we can in $\mathcal{O}(n/\log_\sigma n + P_t(k,\ell,\sigma))$ time and $\mathcal{O}(n/\log_\sigma n + P_s(k,\ell,\sigma))$ working space build a data structure of size $\mathcal{O}(n/\log_\sigma n + S(k,\ell,\sigma))$ that supports the following queries:

- *Given any position $j \in [1 \mathinner{.\,.} n]$ and any $\ell \in [1 \mathinner{.\,.} n+1-j]$, in $\mathcal{O}(\log\log n + Q(k,\ell,\sigma))$ time compute the position $\min \mathrm{Occ}(P,T)$, where $P = T[j \mathinner{.\,.} j+\ell)$.*
- *Given the packed representation[7] of any pattern $P \in [0 \mathinner{.\,.} \sigma)^m$ that satisfies $\mathrm{Occ}(P,T) \neq \emptyset$, in $\mathcal{O}(\log\log n + Q(k,\ell,\sigma) + m/\log_\sigma n)$ time compute the position $\min \mathrm{Occ}(P,T)$.*

b) Related Work: Ellert [103] described an $\mathcal{O}(n/\log_\sigma n)$-time algorithm that computes a 3-approximation of LZ77. Two $\mathcal{O}(z)$-working-space algorithms constructing a 2-approximation and a $(1+\epsilon)$-approximation running in $\mathcal{O}(n\log n)$ and

[7]By a “packed” representation of a string $S \in [0 \mathinner{.\,.} \sigma)^m$, we mean its $\mathcal{O}(m/\log_\sigma n)$-space encoding in memory; see Section II.

$\mathcal{O}(n\log^2 n)$ time, respectively, were proposed in [109]. A practical external-memory approximation was described in [79].

Rightmost LZ77 is a variant of LZ77 where the encoding of every phrase refers to its rightmost previous occurrence. An $\mathcal{O}(n\log n)$-time and $\mathcal{O}(n)$-space algorithm for this problem was given in [110], [111]. This has been improved to $\mathcal{O}(n + (n\log\sigma)/\log\log n)$ time and $\mathcal{O}(n)$ space in [112], and further to $\mathcal{O}(n\log\log\sigma + (n\log\sigma)/\sqrt{\log n})$ (deterministic) or $\mathcal{O}(n + (n\log\sigma)/\sqrt{\log n})$ (randomized) time, while using the optimal $\mathcal{O}(n/\log_\sigma n)$ space, in [99]. An $(1+\epsilon)$-approximation of the rightmost LZ77 can be constructed in $\mathcal{O}(n(\log z + \log\log n))$ time and $\mathcal{O}(n)$ space; see [113].

A variant of LZ77 that requires phrases to have earlier occurrences ending at phrase boundaries is called *LZ-End* [50]. This variant was proved to achieve an approximation ratio of $\mathcal{O}(\log^2(n/z))$ in [13]. This was recently improved by a factor $\Theta(\log\log(n/z))$ [114]. Algorithms computing LZ-End in $\mathcal{O}(n\ell_{\max}(\log\sigma + \log\log n))$ time and $\mathcal{O}(n)$ space, or in $\mathcal{O}(n\ell_{\max}\log^{1+\epsilon} n)$ time and $\mathcal{O}(n/\log_\sigma n)$ space (where $\ell_{\max}$ is the length of the longest phrase and $\epsilon > 0$ is any positive constant) were given in [20]. A construction running in $\mathcal{O}(n)$ time and space was then given in [115]. The same time and space were achieved for the rightmost variant of LZ-End in [116]. Lastly, an LZ-End factorization algorithm running in $\mathcal{O}(n\log\ell_{\max})$ expected time and $\mathcal{O}(z_{\text{end}} + \ell_{\max})$ working space (where z_{end} is the number of phrases in the LZ-End factorization) was given in [117].

II. Preliminaries

a) Basic Definitions: A *string* is a finite sequence of characters from a given *alphabet* Σ. The length of a string S is denoted $|S|$. For $i \in [1\mathinner{.\,.}|S|]$, the ith character of S is denoted $S[i]$. A *substring* of S is a string of the form $S[i\mathinner{.\,.}j) = S[i]S[i+1]\cdots S[j-1]$ for some indices $1 \le i \le j \le |S|+1$. Substrings of the form $S[1\mathinner{.\,.}j)$ and $S[i\mathinner{.\,.}|S|{+}1)$ are called *prefixes* and *suffixes*, respectively. We use $\overline{S}$ to denote the *reverse* of S, i.e., $S[|S|]\cdots S[2]S[1]$. We denote the *concatenation* of two strings U and V, that is, $U[1]\cdots U[|U|]V[1]\cdots V[|V|]$, by UV or $U\cdot V$. Furthermore, $S^k = \bigodot_{i=1}^{k} S$ is the concatenation of $k \in \mathbb{Z}_{\ge 0}$ copies of S; note that $S^0 = \varepsilon$ is the *empty string*. A nonempty string S is said to be *primitive* if it cannot be written as $S = U^k$, where $k \ge 2$. An integer $p \in [1\mathinner{.\,.}|S|]$ is a *period* of S if $S[i] = S[i+p]$ holds for every $i \in [1\mathinner{.\,.}|S|-p]$. We denote the shortest period of S as $\operatorname{per}(S)$. For every $S \in \Sigma^+$, we define the infinite power S^∞ so that $S^\infty[i] = S[1 + (i-1) \bmod |S|]$ for $i \in \mathbb{Z}$. In particular, $S = S^\infty[1\mathinner{.\,.}|S|]$.

By $\operatorname{lcp}(U,V)$ we denote the length of the longest common prefix of strings U and V. For any string $S \in \Sigma^*$ and any $j_1, j_2 \in [1\mathinner{.\,.}|S|+1]$, we denote $\mathrm{LCE}_S(j_1,j_2) = \operatorname{lcp}(S[j_1\mathinner{.\,.}|S|], S[j_2\mathinner{.\,.}|S|])$. We use $\preceq$ to denote the order on Σ, extended to the *lexicographic* order on Σ^* so that $U, V \in \Sigma^*$ satisfy $U \preceq V$ if and only if either (a) U is a prefix of V, or (b) $U[1\mathinner{.\,.}i) = V[1\mathinner{.\,.}i)$ and $U[i] \prec V[i]$ holds for some $i \in [1\mathinner{.\,.}\min(|U|,|V|)]$.

i	$\mathrm{SA}_T[i]$	$T[\mathrm{SA}_T[i]\mathinner{.\,.}n]$
1	19	a
2	14	aababa
3	5	aababababaababa
4	17	aba
5	12	abaababa
6	3	abaababababaababa
7	15	ababa
8	10	ababaababa
9	8	abababaababa
10	6	ababababaababa
11	18	ba
12	13	baababa
13	4	baababababaababa
14	16	baba
15	11	babaababa
16	2	babaababababaababa
17	9	bababaababa
18	7	babababaababa
19	1	bbabaababababaababa

Fig. 1. A list of all sorted suffixes of $T = \texttt{bbabaababababaababa}$ along with the suffix array.

b) Suffix Array: For any string $T \in \Sigma^n$ (of length $n \ge 1$), the *suffix array* $\mathrm{SA}_T[1\mathinner{.\,.}n]$ of T is a permutation of $[1\mathinner{.\,.}n]$ such that $T[\mathrm{SA}_T[1]\mathinner{.\,.}n] \prec T[\mathrm{SA}_T[2]\mathinner{.\,.}n] \prec \cdots \prec T[\mathrm{SA}_T[n]\mathinner{.\,.}n]$, i.e., $\mathrm{SA}_T[i]$ is the starting position of the lexicographically ith suffix of T; see Fig. 1 for an example. The *inverse suffix array* $\mathrm{ISA}_T[1\mathinner{.\,.}n]$ (also denoted $\mathrm{SA}_T^{-1}[1\mathinner{.\,.}n]$) is the inverse permutation of SA_T, i.e., $\mathrm{ISA}_T[j] = i$ holds if and only if $\mathrm{SA}_T[i] = j$. Intuitively, $\mathrm{ISA}_T[j]$ stores the lexicographic *rank* of $T[j\mathinner{.\,.}n]$ among the suffixes of T.

Definition II.1. For any $T \in \Sigma^n$ and $P \in \Sigma^*$, we define

$$\begin{aligned}\mathrm{Occ}(P,T) &= \{j \in [1\mathinner{.\,.}n] : j + |P| \le n+1 \text{ and } \\ &\qquad\qquad T[j\mathinner{.\,.}j+|P|) = P\},\\ \mathrm{RangeBeg}(P,T) &= |\{j \in [1\mathinner{.\,.}n] : T[j\mathinner{.\,.}n] \prec P\}|,\\ \mathrm{RangeEnd}(P,T) &= \mathrm{RangeBeg}(P,T) + |\mathrm{Occ}(P,T)|.\end{aligned}$$

In other words, $\mathrm{Occ}(P,T)$ consists of the starting positions of the (exact) occurrences of P in T, with the convention that $\mathrm{Occ}(\varepsilon,T) = [1\mathinner{.\,.}n]$ holds if $n = |T| > 0$. The two values $\mathrm{RangeBeg}(P,T)$ and $\mathrm{RangeEnd}(P,T)$ are defined to be endpoints of the so-called *SA-interval* representing the occurrences of P in T. Formally, $\mathrm{Occ}(P,T) = \{\mathrm{SA}_T[i] : i \in (\mathrm{RangeBeg}(P,T)\mathinner{.\,.}\mathrm{RangeEnd}(P,T)]\}$ holds for every pattern $P \in \Sigma^*$, including when $P = \varepsilon$ and when $\mathrm{Occ}(P,T) = \emptyset$.

c) Lempel–Ziv Compression: A fragment $T[j\mathinner{.\,.}j+\ell)$ of T is a *previous factor* if it has an earlier occurrence in T, i.e., $\mathrm{LCE}_T(i,j) \ge \ell$ holds for some $i \in [1\mathinner{.\,.}j)$. An *LZ77-like factorization* of T is a decomposition $T = f_1 \cdots f_z$ into non-empty *phrases* such that each phrase f_k with $|f_k| > 1$ is a previous factor. In the underlying *LZ77-like representation*, every phrase $f_k = T[j\mathinner{.\,.}j+\ell)$ that is a previous factor is encoded as (i,ℓ), where $i \in [1\mathinner{.\,.}j)$ satisfies $\mathrm{LCE}_T(i,j) \ge \ell$ (and is chosen arbitrarily in case of multiple options); if $f_k = T[j]$ is not a previous factor, we encode it as $(T[j], 0)$.

The LZ77 factorization [3] of a string T is then just an LZ77-like factorization constructed by greedily factorizing T from left to right into the longest possible phrases. More precisely, the kth phrase f_k is the longest previous factor starting at position $1 + |f_1 \cdots f_{k-1}|$; if no previous factor starts there, then f_k consists of a single character. This greedy construction yields the smallest LZ77-like factorization of T [2, Theorem 1]. We denote the number of phrases in the LZ77 factorization of T by $z(T)$.

For example, the text of Fig. 1 has LZ77 factorization $T = \mathtt{b} \cdot \mathtt{b} \cdot \mathtt{a} \cdot \mathtt{ba} \cdot \mathtt{aba} \cdot \mathtt{bababa} \cdot \mathtt{ababa}$ with $z(T) = 7$ phrases, and the underlying LZ77 representation is $(\mathtt{b}, 0), (1, 1), (\mathtt{a}, 0), (2, 2), (3, 3), (7, 6), (10, 5)$.

A variant of LZ77 factorization in which we additionally require that the earlier occurrence of every phrase does not overlap the phrase itself is called the *non-overlapping LZ77*. We denote the number of phrases in this variant by $z_{\text{no}}(T)$. The non-overlapping LZ77 factorization of the text of Fig. 1 is $T = \mathtt{b} \cdot \mathtt{b} \cdot \mathtt{a} \cdot \mathtt{ba} \cdot \mathtt{aba} \cdot \mathtt{baba} \cdot \mathtt{baababa}$ with $z_{\text{no}}(T) = 7$ phrases.

d) String Synchronizing Sets: String synchronizing sets [107] allow for a locally-consistent selection of positions in a given text T. The underlying parameter τ governs the context size (with respect to which the selection is consistent) and the achievable size of the synchronizing set.

Definition II.2 (τ-synchronizing set [107])**.** Let $T \in \Sigma^n$ be a string and let $\tau \in [1 \mathinner{.\,.} \lfloor \frac{n}{2} \rfloor]$ be a parameter. A set $\mathsf{S} \subseteq [1 \mathinner{.\,.} n - 2\tau + 1]$ is called a *τ-synchronizing set* of T if it satisfies the following *consistency* and *density* conditions:

1) If $T[i \mathinner{.\,.} i + 2\tau) = T[j \mathinner{.\,.} j + 2\tau)$, then $i \in \mathsf{S}$ holds if and only if $j \in \mathsf{S}$ (for $i, j \in [1 \mathinner{.\,.} n - 2\tau + 1]$),
2) $\mathsf{S} \cap [i \mathinner{.\,.} i + \tau) = \emptyset$ if and only if $i \in \mathsf{R}(\tau, T)$ (for $i \in [1 \mathinner{.\,.} n - 3\tau + 2]$), where

$$\mathsf{R}(\tau, T) := \{i \in [1 \mathinner{.\,.} n-3\tau+2] : \operatorname{per}(T[i \mathinner{.\,.} i+3\tau-2]) \le \tfrac{1}{3}\tau\}.$$

Remark II.3. In most applications, we want to minimize $|\mathsf{S}|$. Note, however, that the density condition imposes a lower bound $|\mathsf{S}| = \Omega(\frac{n}{\tau})$ for strings of length $n \ge 3\tau - 1$ that do not contain substrings of length $3\tau - 1$ with period at most $\frac{1}{3}\tau$. Thus, we cannot hope to achieve an upper bound improving in the worst case upon the following ones.

Theorem II.4 ([107, Proposition 8.10])**.** *For every string T of length n and parameter $\tau \in [1 \mathinner{.\,.} \lfloor \frac{n}{2} \rfloor]$, there exists a τ-synchronizing set S of size $|\mathsf{S}| = \mathcal{O}\left(\frac{n}{\tau}\right)$. Moreover, if $T \in [0 \mathinner{.\,.} \sigma)^n$, where $\sigma = n^{\mathcal{O}(1)}$, such S can be deterministically constructed in $\mathcal{O}(n)$ time.*

Theorem II.5 ([107, Theorem 8.11])**.** *For every constant $\mu < \frac{1}{5}$, given the packed representation of a string $T \in [0 \mathinner{.\,.} \sigma)^n$ and a positive integer $\tau \le \mu \log_\sigma n$, one can deterministically construct in $\mathcal{O}(\frac{n}{\tau})$ time a τ-synchronizing set of size $\mathcal{O}(\frac{n}{\tau})$.*

e) Rank and Selection Queries:

Definition II.6. For a string $S \in \Sigma^n$, we define:

Rank query $\mathsf{rank}_{S,a}(j)$: Given a symbol $a \in \Sigma$ and a position $j \in [0 \mathinner{.\,.} n]$, compute $|\{i \in [1 \mathinner{.\,.} j] : S[i] = a\}|$.

Selection query $\mathsf{select}_{S,a}(r)$: Given a symbol $a \in \Sigma$ and an integer $r \in [1 \mathinner{.\,.} \mathsf{rank}_{S,a}(n)]$, find the rth smallest element of $\{i \in [1 \mathinner{.\,.} n] : S[i] = a\}$.

Theorem II.7 (Rank and selection queries in bitvectors [118]–[121])**.** *For every string $S \in \{0, 1\}^*$, there exists a data structure of $\mathcal{O}(|S|)$ bits answering rank and selection queries in $\mathcal{O}(1)$ time. Moreover, given the packed representations of m binary strings of total length n, the data structures for all these strings can be constructed in $\mathcal{O}(m + n/\log n)$ time.*

f) Model of Computation: We use the standard word RAM model of computation [83] with w-bit *machine words*, where $w \ge \log n$, and all standard bit-wise and arithmetic operations taking $\mathcal{O}(1)$ time. Unless explicitly stated otherwise, we measure the space complexity in machine words.

In the RAM model, strings are usually represented as arrays, with each character occupying one memory cell (or a constant number of memory cells if $\sigma > n$). A single character, however, only needs $\lceil \log \sigma \rceil$ bits, which might be much less than w. We can therefore store (the *packed representation* of) a string $S \in [0 \mathinner{.\,.} \sigma)^m$ using $\mathcal{O}(\lceil \frac{m \log \sigma}{w} \rceil)$ words.

III. Technical Overview

Consider a text $T \in [0 \mathinner{.\,.} \sigma)^n$. We assume that $2 \le \sigma < n^{1/7}$; larger alphabet sizes σ satisfy $\log \sigma = \Theta(\log n)$, and, in that case, most of the problems considered in this paper can be solved using standard large-alphabet techniques. For example, whenever $\sigma = n^{\Theta(1)}$, the LZ77 factorization algorithm from [59] runs in $\mathcal{O}(n) = \mathcal{O}(n/\log_\sigma n)$ time and $\mathcal{O}(n) = \mathcal{O}(n/\log_\sigma n)$ space.

A. Index for Leftmost Occurrences

We now outline how, given any constant $\epsilon \in (0, 1)$ along with the $\mathcal{O}(n/\log_\sigma n)$-space representation of $T \in [0 \mathinner{.\,.} \sigma)^n$, in $\mathcal{O}(n \log \sigma/\sqrt{\log n})$ time and $\mathcal{O}(n/\log_\sigma n)$ working space, we can construct an index of size $\mathcal{O}(n/\log_\sigma n)$ that, given any position $j \in [1 \mathinner{.\,.} n]$ and any length $\ell \in [1 \mathinner{.\,.} n - j + 1]$, in $\mathcal{O}(\log^\epsilon n)$ time returns the position $\min \operatorname{Occ}(P, T)$ (see Definition II.1) for $P = T[j \mathinner{.\,.} j + \ell)$. Our index can also compute $\min \operatorname{Occ}(P, T)$ in $\mathcal{O}(\log^\epsilon n + m/\log_\sigma n)$ time given the $\mathcal{O}(m/\log_\sigma n)$-space representation of any $P \in [0 \mathinner{.\,.} \sigma)^m$ satisfying $\operatorname{Occ}(P, T) \ne \emptyset$, which is of independent interest.

a) The Index Core: Let $\mu \in (0, \frac{1}{6})$ be a positive constant such that $\tau := \mu \log_\sigma n$ is a positive integer. Such μ exists because $\log_\sigma n > 7$ follows from $\sigma < n^{1/7}$. Consider integers $j \in [1 \mathinner{.\,.} n]$ and $\ell \in [1 \mathinner{.\,.} n+1-j]$, and a pattern $P \in [0 \mathinner{.\,.} \sigma)^m$ of length $m > 0$ given in $\mathcal{O}(1 + m/\log_\sigma n)$ space.

We begin by observing that the number of strings $X \in [0 \mathinner{.\,.} \sigma)^*$ satisfying $|X| < 3\tau - 1$ is bounded by $\sigma^{3\tau} \le n^{1/2}$. Thus, we can precompute and store $\min \operatorname{Occ}(X, T)$ for all of them. This computation is easily done in $\mathcal{O}(n/\log_\sigma n)$ exploiting the packed representation of T. Using this lookup table, we can answer queries when $\ell < 3\tau - 1$ and $m < 3\tau - 1$.

Let us now assume that $\ell \ge 3\tau - 1$ and $m \ge 3\tau - 1$. The computation of $\min \operatorname{Occ}(T[j \mathinner{.\,.} j + \ell), T)$ works differently, depending on whether $\mathrm{per}(T[j\mathinner{.\,.} j+\ell)) \le \frac{1}{3}\tau$ (or equivalently $j \in \mathsf{R}(\tau,T)$; see Definition II.2). For explicit patterns P, we also proceed depending on whether $\mathrm{per}(P) \le \frac{1}{3}\tau$, in which case we call P *τ-periodic*. To distinguish these cases, we store the packed representation of T (to retrieve $T[j\mathinner{.\,.} j+\ell)$) and a lookup table keeping $\mathrm{per}(X)$ for every $X \in [0\mathinner{.\,.}\sigma)^{3\tau-1}$. Such table takes $\mathcal{O}(\sigma^{3\tau}) = \mathcal{O}(n^{1/2})$ space and is easily computed in $\mathcal{O}(n/\log_\sigma n)$ time.

b) The Nonperiodic Patterns and Positions: Assume that $j \notin \mathsf{R}(\tau,T)$ and P is τ-nonperiodic. Assume that we computed (using Theorem II.5 in $\mathcal{O}(n/\log_\sigma n)$ time) a τ-synchronizing set S of T satisfying $|\mathsf{S}| = \mathcal{O}(\frac{n}{\tau}) = \mathcal{O}(n/\log_\sigma n)$. By the density condition of S (Definition II.2(2)), the successor $\mathrm{succ}_{\mathsf{S}}(j)$ in S of every position $j \in [1\mathinner{.\,.} n-3\tau+2] \setminus \mathsf{R}(\tau,T)$ satisfies $\mathrm{succ}_{\mathsf{S}}(j) - j < \tau$. This implies that the substring $D := T[j\mathinner{.\,.}\mathrm{succ}_{\mathsf{S}}(j)+2\tau)$ (called the *distinguishing prefix* of $T[j\mathinner{.\,.} j+\ell)$) satisfies $|D| \le 3\tau-1$. Moreover, by the consistency condition of S (Definition II.2(1)), every occurrence $j' \in \mathrm{Occ}(D,T)$ satisfies $j'+\delta_{\text{text}} \in \mathsf{S}$, where $\delta_{\text{text}} = |D|-2\tau$.

Observation: Computation of $\min \mathrm{Occ}(T[j\mathinner{.\,.} j+\ell),T)$ *and* $\min\mathrm{Occ}(P,T)$ *can be efficiently reduced to a 4-sided RMQ query.* By the discussion above, we can characterize $\mathrm{Occ}(T[j\mathinner{.\,.} j+\ell),T)$ as a set of all positions of the form $s-\delta_{\text{text}}$, where $s \in \mathsf{S}$ satisfies

(1) $T^\infty[s-\delta_{\text{text}}\mathinner{.\,.} s+2\tau) = D$ (or equivalently, $s-\delta_{\text{text}} \in \mathrm{Occ}(D,T)$), and
(2) $s \in \mathrm{Occ}(T[j+\delta_{\text{text}}\mathinner{.\,.} j+\ell),T)$.

Analogous characterization holds for P. Consider now a sequence $(s_i)_{i\in[1\mathinner{.\,.}|\mathsf{S}|]}$ containing positions in S sorted according to the lexicographical order of the corresponding suffixes. This sequence can be constructed in $\mathcal{O}(n/\log_\sigma n)$ time (see the full version [1]). Given a pair (j,ℓ) (resp. $\mathcal{O}(m/\log_\sigma n)$-space representation of P), we can in $\mathcal{O}(\log\log n)$ (resp. $\mathcal{O}(\log\log n + m/\log_\sigma n)$) time compute the boundaries of a range $(b\mathinner{.\,.} e]$ such that $(s_i)_{i\in(b\mathinner{.\,.} e]}$ consists of all positions in the set $\mathsf{S}\cap\mathrm{Occ}(T[j+\delta_{\text{text}}\mathinner{.\,.} j+\ell),T)$ (resp. $\mathsf{S}\cap\mathrm{Occ}(P[1+\delta_{\text{text}}\mathinner{.\,.} m],T)$). This is easily achieved using weighted ancestor queries on a compact trie of suffixes starting in S. Then, $\{s_i\}_{i\in(b\mathinner{.\,.} e]}$ is the set of all $s \in \mathsf{S}$ satisfying condition (2). To satisfy condition (1), we need to select all $s \in \mathsf{S}$ for which $s-\delta_{\text{text}} \in \mathrm{Occ}(D,T)$. If each position $s_i \in \mathsf{S}$ is represented as a point (i, D_i), where $\overline{D_i} = T^\infty[s-\tau\mathinner{.\,.} s+2\tau)$, then the sought subset consists of all $s_i \in \mathsf{S}$ such that $\overline{D}$ is a prefix of D_i, which is equivalent to $\overline{D} \preceq D_i \prec \overline{D}c^\infty$, where $c = \sigma-1$. Thus, finding $\min\mathrm{Occ}(T[j\mathinner{.\,.} j+\ell),T)$ (resp. $\min\mathrm{Occ}(P,T)$) reduces to the 4-sided range RMQ query in a rectangle obtained by intersecting ranges $(b\mathinner{.\,.} e]$ and $[\overline{D}, \overline{D}c^\infty)$.

The bottleneck of the above reduction is answering the orthogonal RMQ queries because it would require a data structure supporting 4-sided RMQ in $\mathcal{O}(\log^\epsilon n)$ time. The only such structure [108] is not known to admit a sufficiently fast construction algorithm. Thus, we take a different approach by directly answering the query characterizing the set $\mathrm{Occ}(T[j\mathinner{.\,.} j+\ell),T)$ and $\mathrm{Occ}(P,T)$. As noted above, $T^\infty[s_i-\delta_{\text{text}}\mathinner{.\,.} s_i+2\tau) = D$ holds if and only if $\overline{D}$ is a prefix of D_i. Taking into account the constraint $i \in (b\mathinner{.\,.} e]$ on the other axis, we obtain a *prefix range minimum query*, formally defined as follows:

Definition III.1 (Prefix RMQ)**.** Let $A \in \mathbb{Z}^m_{\ge 0}$ be a sequence of m nonnegative integers and $S \in (\Sigma^*)^m$ be a sequence of m strings over alphabet Σ. For every $b,e \in [0\mathinner{.\,.} m]$ and $X \in \Sigma^*$ we define

$$\mathsf{prefix\text{-}rmq}_{A,S}(b,e,X) := \arg\min\{A[i] : i \in (b\mathinner{.\,.} e] \text{ and } X \text{ is a prefix of } S[i]\}.$$

We assume that $\mathsf{prefix\text{-}rmq}_{A,S}(b,e,X) = \arg\min\emptyset = \infty$ if there is no $i \in (b\mathinner{.\,.} e]$ for which X is a prefix of $S[i]$.

Thus, with $A_{\mathsf{S}}[1\mathinner{.\,.} n']$ and $A_{\text{str}}[1\mathinner{.\,.} n']$ defined as arrays of length $n' = |\mathsf{S}|$ such that $A_{\mathsf{S}}[i] = s_i$ and $A_{\text{str}}[i] = \overline{D_i}$, it holds $\min\mathrm{Occ}(P,T) = A_{\mathsf{S}}[\mathsf{prefix\text{-}rmq}_{A_{\mathsf{S}},A_{\text{str}}}(b,e,\overline{D})] - \delta_{\text{text}}$.

To our knowledge, prefix range minimum queries have not been studied before. We develop a solution that not only answers these queries in $\mathcal{O}(\log^\epsilon n)$ time but also admits efficient construction, as described in the following theorem proved in the full version [1] of this paper. In addition to solving a novel type of query, this requires improvements of existing structures for standard RMQ.

Theorem III.2. *For all integers $h,m,\ell,\sigma \in \mathbb{Z}_{>0}$ satisfying $h \ge 2$ and $m \ge \sigma^\ell \ge 2$, and for all equal-length sequences $A \in [0\mathinner{.\,.} m)^{\le m}$ and $S \in ([0\mathinner{.\,.}\sigma)^\ell)^{\le m}$, there exists a data structure of size $\mathcal{O}(m\log_h(h\ell))$ that answers prefix RMQ queries in $\mathcal{O}(h\log\log m\log_h(h\ell))$ time. Moreover, it can be constructed in $\mathcal{O}(m\min(\ell,\sqrt{\log m})\log_h(h\ell))$ time using $\mathcal{O}(m\log_h(h\ell))$ space assuming that S is given in the packed representation.*

Applied to arrays $A_{\mathsf{S}}[1\mathinner{.\,.} n']$ and $A_{\text{str}}[1\mathinner{.\,.} n']$, Theorem III.2 yields a data structure of size $\mathcal{O}(n/\log_\sigma n)$ that answers prefix RMQ in $\mathcal{O}(\log^\epsilon n)$ time, and can be constructed in $\mathcal{O}((n\log\sigma)/\sqrt{\log n})$ time and $\mathcal{O}(n/\log_\sigma n)$ working space. We remark, however, that other tradeoffs for prefix RMQ will automatically yield new tradeoffs for indexes for leftmost occurrences, and hence also new LPF/LPnF indexes and LZ77 factorization algorithms.

c) The Periodic Patterns and Positions: Let us now assume that $j \in \mathsf{R}(\tau,T)$ and P is τ-periodic. First, observe that $T[j\mathinner{.\,.} j+\ell)$ and P are both prefixed with a string $X \in [0\mathinner{.\,.}\sigma)^{3\tau-1}$ satisfying $\mathrm{per}(X) \le \frac{1}{3}\tau$. Thus, $\mathrm{Occ}(T[j\mathinner{.\,.} j+\ell),T), \mathrm{Occ}(P,T) \subseteq \mathsf{R}(\tau,T)$ holds by Definition II.2. The central property of $\mathsf{R}(\tau,T)$ is that every maximal block of positions in $\mathsf{R}(\tau,T)$ corresponds to a *τ-run*, i.e., a maximal fragment Y of T satisfying $|Y| \ge 3\tau-1$ and $\mathrm{per}(Y) \le \frac{1}{3}\tau$. The gap between $|Y|$ and $\mathrm{per}(Y)$ ensures that τ-runs overlap by fewer than $\frac{2}{3}\tau$ symbols, so the number of τ-runs is $\mathcal{O}(\frac{n}{\tau}) = \mathcal{O}(n/\log_\sigma n)$.

To efficiently process τ-runs, we introduce the following definitions. Let $x \in \mathsf{R}(\tau,T)$, and let $T[y]$ be the position immediately following of the τ-run containing $T[x \mathinner{.\,.} x+3\tau-1)$. By $y-x \geq 3\tau-1$ and $p := \mathrm{per}(T[x \mathinner{.\,.} y)) \leq \frac{1}{3}\tau$, we can uniquely write $T[x \mathinner{.\,.} y) = H'H^kH''$, where $H = \min\{T[x+\delta \mathinner{.\,.} x+\delta+p) : \delta \in [0 \mathinner{.\,.} p)\}$ is the so-called *Lyndon root* of the run and H' (resp. H'') is a proper suffix (resp. prefix) of H. We denote $e(x,\tau,T) = y$, $\mathrm{root}(x,\tau,T) = H$, and $e^{\mathrm{full}}(x,\tau,T) = y - |H''|$. We also let $\mathrm{type}(x,\tau,T) = -1$ if $T[y] \prec T[y-p]$ and $\mathrm{type}(x,\tau,T) = +1$ otherwise. The above definitions naturally generalize to τ-periodic patterns. Let us focus on the computation of $\min \mathrm{Occ}(P,T)$ (the computation of $\min \mathrm{Occ}(T[j \mathinner{.\,.} j+\ell),T)$ proceeds similarly). The query algorithm differs depending on whether the periodic prefix of P ends before $|P|$, i.e., $e(P,\tau) \leq |P|$. In that case, we call P *partially periodic*. Otherwise, i.e., when $e(P,\tau) = |P|+1$, we call P *fully periodic*.

Observation 1: Computation of $\min \mathrm{Occ}(P,T)$ *in the partially periodic case can be efficiently reduced to a 3-sided RMQ query.* Observe that if $e(P,\tau) \leq |P|$, then the end of the periodic prefix of P has to align with the end of a periodic τ-run in T with the same root, i.e., if $j \in \mathrm{Occ}(P,T)$, then $j \in \mathsf{R}(\tau,T)$, $\mathrm{root}(j,\tau,T) = \mathrm{root}(P,\tau)$, and $e(j,\tau,T) - j = e(P,\tau) - 1$. At the same time, the remaining suffix of P must follow the run in the text, i.e., we need to have $j + \delta_{\mathrm{text}} \in \mathrm{Occ}(P',T)$, where $\delta_{\mathrm{text}} = e(P,\tau) - 1$ and $P' = P(\delta_{\mathrm{text}} \mathinner{.\,.} |P|]$. If we sorted every τ-run $T[x \mathinner{.\,.} y)$ first according to its root and then according to $T[y \mathinner{.\,.} n]$, then runs satisfying the second criterion would form a range. This, however, would make it difficult to make sure the end of the periodic substring in the text is properly aligned with the periodic prefix of P solely based on the length of the τ-run. Sorting every τ-run $T[x \mathinner{.\,.} y)$ first according to $\mathrm{root}(x,\tau,T)$ and then according to the suffix $T[e^{\mathrm{full}}(x,\tau,T) \mathinner{.\,.} n]$ solves the alignment problem. Identifying the range in the resulting sorted sequence of τ-runs then again reduces to a weighted ancestor query. To simultaneously also align according to the root, we slightly back away in both the pattern and the text by a multiple of $|\mathrm{root}(P,\tau)|$. After identifying the range containing τ-runs with the right-context matching P, it remains to only select runs $T[x \mathinner{.\,.} y)$ whose length is at least the length of the periodic prefix of P. Then, $\min \mathrm{Occ}(P,T)$ is located in the leftmost selected τ-run. This corresponds to a 3-sided RMQ query. To answer it efficiently, we exploit the fact that both the maximum and the sum of the y-coordinates of the corresponding point-set are bounded by n.

Observation 2: Computation of $\min \mathrm{Occ}(P,T)$ *in the fully periodic case can be efficiently reduced to a rank query on a bitvector.* Let us denote

$$\mathsf{R}^-(\tau,T) := \{j \in \mathsf{R}(\tau,T) : \mathrm{type}(j,\tau,T) = -1\}$$

and focus on computing $\min \mathrm{Occ}(P,T) \cap \mathsf{R}^-(\tau,T)$ (the other minimum is computed analogously). Let $\mathsf{R}^-_{\min}(\tau,T)$ be the set of all $j \in \mathsf{R}^-(\tau,T)$ satisfying $j = \min \mathrm{Occ}(T[j \mathinner{.\,.} e(j,\tau,T)),T) \cap \mathsf{R}^-(\tau,T)$. The key idea is to store a bitvector marking all $i \in [1 \mathinner{.\,.} n]$ such that $\mathrm{SA}_T[i] \in \mathsf{R}^-_{\min}(\tau,T)$. Because the values $e(j,\tau,T) - j$ increase within every block of SA_T entries containing all $j \in \mathsf{R}^-(\tau,T) \cap \mathrm{Occ}(P[1 \mathinner{.\,.} 3\tau-1],T)$, it follows that, given the SA_T-range corresponding to P, the computation of $\min \mathrm{Occ}(P,T)$ reduces to a rank/select query on the above bitvector. The main challenge is thus computing the bitvector. This step is one of the most technically challenging parts of our data structure. The construction is a complex algorithm that first prepares the set of "events" and then computes partial bitvectors using the sweeping technique. This requires developing numerous new combinatorial results and efficient solutions for offline range counting and dynamic one-sided RMQ; see the full version for details [1]. Combining all these ingredients, we achieve the optimal $\mathcal{O}(n/\log_\sigma n)$-time construction.

d) Summary of New Techniques: Our key technical contributions can be summarized as follows:

- We define a new query called *prefix RMQ* and develop an efficient solution, improving the construction of small-alphabet RMQ on the way to this result.
- We show how to use the above to find leftmost occurrences of nonperiodic patterns.
- To efficiently handle periodic patterns, we prove numerous new combinatorial results characterizing leftmost occurrences of substrings, and we develop efficient solutions for offline range counting, three-sided RMQ, and dynamic one-sided RMQ.
- Using the above queries, we show how to compute leftmost occurrences of periodic patterns; combined with the above result, this gives an optimal-size index constructible in $\mathcal{O}((n\log\sigma)/\sqrt{\log n})$ time and $\mathcal{O}(n/\log_\sigma n)$ working space. This reduction is very efficient and depends almost entirely on the tradeoff for prefix RMQ queries.
- Using the above index, we design a structure that provides random access to the LPF and LPnF arrays (see Section III-B).

Putting everything together, we get the first $o(n)$-time LZ77 factorization after nearly 50 years.

B. Index for Longest Previous Factors

We outline how, given any constant $\epsilon \in (0,1)$ along with the $\mathcal{O}(n/\log_\sigma n)$-space representation of $T \in [0 \mathinner{.\,.} \sigma)^n$, in $\mathcal{O}(n\log\sigma/\sqrt{\log n})$ time and $\mathcal{O}(n/\log_\sigma n)$ working space, we can construct an index of size $\mathcal{O}(n/\log_\sigma n)$ that, given $i \in [1 \mathinner{.\,.} n]$, returns $\mathrm{LPF}_T[i]$ and $\mathrm{LPFMinOcc}_T[i]$. For simplicity, we focus on the LPF array allowing self-overlaps; the computations for the non-overlapping variant are similar.

The arrays $\mathrm{LPF}_T[1 \mathinner{.\,.} n]$ and $\mathrm{LPFMinOcc}_T[1 \mathinner{.\,.} n]$ are formally defined as follows. First, we let $\mathrm{LPF}_T[1] = 0$. For every $i \in [2 \mathinner{.\,.} n]$, the value $\mathrm{LPF}_T[i]$ is the maximal $\ell \geq 0$ such that $\min \mathrm{Occ}(T[i \mathinner{.\,.} i+\ell),T) < i$. The entry $\mathrm{LPFMinOcc}_T[i]$ contains either $T[i]$ (if $\mathrm{LPF}_T[i] = 0$) or $\min \mathrm{Occ}(T[i \mathinner{.\,.} i+\mathrm{LPF}_T[i]),T)$ (otherwise).

We begin by observing that $\mathrm{LPF}_T[i] \geq \mathrm{LPF}_T[i-1] - 1$ holds for every $i \in [2\,.\,.\,n]$. This implies that, given any indexes $i, k \in [1\,.\,.\,n]$ satisfying $i \leq k$, we can use the values $\mathrm{LPF}_T[i]$ and $\mathrm{LPF}_T[k]$ to compute a common lower and upper bound for $\mathrm{LPF}_T[j]$ with $j \in [i\,.\,.\,k]$. More precisely, we can then compute $\ell_{\min}$ and $\ell_{\max}$ such that $\mathrm{LPF}_T[j] \in [\ell_{\min}\,.\,.\,\ell_{\max}]$ and $\ell_{\max} - \ell_{\min} = \mathcal{O}((\mathrm{LPF}_T[k] - \mathrm{LPF}_T[i]) + (k-i))$.

Let $b = \Theta(\log^3 n)$ and $b' = \Theta(\log^6 n)$ be integers. We partition LPF_T into blocks of size b and store the $\mathrm{LPF}_T[\cdot]$ values at all block boundaries. Additionally, every block $\mathrm{LPF}_T((i-1)b\,.\,.\,ib]$ satisfying $\mathrm{LPF}_T[ib] - \mathrm{LPF}_T[(i-1)b] \geq b' - b$ has all the $\mathrm{LPF}_T[\cdot]$ values explicitly stored in our data structure, and the index i of the block is marked in a bitvector. At query time, we first check if the queried position x is in one of the stored blocks. If so, we have the answer. Otherwise, utilizing the above observation, we compute $\ell_{\min}$ and $\ell_{\max}$, and then determine $\mathrm{LPF}_T[j]$ using binary search and the index for leftmost occurrences (constructed for $\epsilon' = \epsilon/2$). Due to our choice of b and b', this takes $\mathcal{O}(\log^{\epsilon'} n \cdot \log(\ell_{\max} - \ell_{\min} + 1)) = \mathcal{O}(\log^{\epsilon/2} n \cdot \log b') = \mathcal{O}(\log^{\epsilon/2} n \cdot \log(\log^6 n)) = \mathcal{O}(\log^{\epsilon} n)$ time. Once the value $\mathrm{LPF}_T[x]$ is computed, we obtain $\mathrm{LPFMinOcc}_T[x]$ in $\mathcal{O}(\log^{\epsilon'} n)$ time.

Once the index for leftmost occurrences is constructed, computing the $\mathrm{LPF}_T[\cdot]$ values at all block boundaries costs $\mathcal{O}(n/b \cdot \log^{1+\epsilon'} n) = o(n/\log_\sigma n)$ time. It remains to bound the number of blocks $\mathrm{LPF}_T((i-1)b\,.\,.\,ib]$ with $\mathrm{LPF}_T[ib] - \mathrm{LPF}_T[(i-1)b] \geq b' - b$ and the total time to process them. For this, observe that the function $f(i) = i + \mathrm{LPF}_T[i]$ is nondecreasing and does not exceed $n+1$. This way, we can bound the number of above blocks by $\mathcal{O}(n/b') = \mathcal{O}(n/\log^6 n)$. Computing all the $\mathrm{LPF}_T[\cdot]$ values in these blocks takes $\mathcal{O}((n/b') \cdot b \cdot \log^{1+\epsilon'} n) = \mathcal{O}(n/\log^3 n \cdot \log^{1+\epsilon'} n) = o(n/\log_\sigma n)$ time. The index construction is thus dominated by the time to build the index for leftmost occurrences.

References

[1] D. Kempa and T. Kociumaka, "Lempel-Ziv (LZ77) factorization in sublinear time," 2024, full version. [Online]. Available: https://arxiv.org/abs/2409.12146

[2] A. Lempel and J. Ziv, "On the complexity of finite sequences," *IEEE Transactions on Information Theory*, vol. 22, no. 1, pp. 75–81, 1976. [Online]. Available: https://doi.org/10.1109/TIT.1976.1055501

[3] J. Ziv and A. Lempel, "A universal algorithm for sequential data compression," *IEEE Transactions on Information Theory*, vol. 23, no. 3, pp. 337–343, 1977. [Online]. Available: https://doi.org/10.1109/TIT.1977.1055714

[4] IEEE, "Milestones: Lempel-Ziv Data Compression Algorithm, 1977," 2004. [Online]. Available: https://ethw.org/Milestones:Lempel-Ziv_Data_Compression_Algorithm,_1977

[5] ——, "Recipients of IEEE Medal of Honor," 2021. [Online]. Available: https://corporate-awards.ieee.org/recipients/ieee-medal-of-honor-recipients/

[6] J. Ziv and A. Lempel, "Compression of individual sequences via variable-rate coding," *IEEE Transactions on Information Theory*, vol. 24, no. 5, pp. 530–536, 1978. [Online]. Available: https://doi.org/10.1109/TIT.1978.1055934

[7] M. Mahoney, "Large Text Compression Benchmark," accessed: 2024-03-20. [Online]. Available: http://mattmahoney.net/dc/text.html

[8] J. Alakuijala, A. Farruggia, P. Ferragina, E. Kliuchnikov, R. Obryk, Z. Szabadka, and L. Vandevenne, "Brotli: A general-purpose data compressor," *ACM Transactions on Information Systems*, vol. 37, no. 1, 2018. [Online]. Available: https://doi.org/10.1145/3231935

[9] W. Rytter, "Application of Lempel–Ziv factorization to the approximation of grammar-based compression," *Theoretical Computer Science*, vol. 302, no. 1–3, pp. 211–222, 2003. [Online]. Available: https://doi.org/10.1016/S0304-3975(02)00777-6

[10] P. Bille, G. M. Landau, R. Raman, K. Sadakane, S. R. Satti, and O. Weimann, "Random access to grammar-compressed strings and trees," *SIAM Journal on Computing*, vol. 44, no. 3, pp. 513–539, 2015. [Online]. Available: https://doi.org/10.1137/130936889

[11] M. Ganardi, A. Jeż, and M. Lohrey, "Balancing straight-line programs," *Journal of the ACM*, vol. 68, no. 4, pp. 27:1–27:40, 2021. [Online]. Available: https://doi.org/10.1145/3457389

[12] D. Belazzougui, M. Cáceres, T. Gagie, P. Gawrychowski, J. Kärkkäinen, G. Navarro, A. O. Pereira, S. J. Puglisi, and Y. Tabei, "Block trees," *Journal of Computer and System Sciences*, vol. 117, pp. 1–22, 2021. [Online]. Available: https://doi.org/10.1016/j.jcss.2020.11.002

[13] D. Kempa and B. Saha, "An upper bound and linear-space queries on the LZ-end parsing," in *33rd Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2022*, J. S. Naor and N. Buchbinder, Eds. SIAM, 2022, pp. 2847–2866. [Online]. Available: https://doi.org/10.1137/1.9781611977073.111

[14] T. Nishimoto, T. I, S. Inenaga, H. Bannai, and M. Takeda, "Fully dynamic data structure for LCE queries in compressed space," in *41st International Symposium on Mathematical Foundations of Computer Science, MFCS 2016*, ser. LIPIcs, P. Faliszewski, A. Muscholl, and R. Niedermeier, Eds., vol. 58. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2016, pp. 72:1–72:15. [Online]. Available: https://doi.org/10.4230/LIPIcs.MFCS.2016.72

[15] T. I, "Longest common extensions with recompression," in *28th Annual Symposium on Combinatorial Pattern Matching, CPM 2017*, ser. LIPIcs, J. Kärkkäinen, J. Radoszewski, and W. Rytter, Eds., vol. 78. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2017, pp. 18:1–18:15. [Online]. Available: https://doi.org/10.4230/LIPIcs.CPM.2017.18

[16] P. Gawrychowski, A. Karczmarz, T. Kociumaka, J. Łącki, and P. Sankowski, "Optimal dynamic strings," in *29th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018*, A. Czumaj, Ed. SIAM, 2018, pp. 1509–1528. [Online]. Available: https://doi.org/10.1137/1.9781611975031.99

[17] A. O. Pereira, G. Navarro, and N. R. Brisaboa, "Grammar compressed sequences with rank/select support," *Journal of Discrete Algorithms*, vol. 43, pp. 54–71, 2017. [Online]. Available: https://doi.org/10.1016/j.jda.2016.10.001

[18] N. Prezza, "Optimal rank and select queries on dictionary-compressed text," in *30th Annual Symposium on Combinatorial Pattern Matching, CPM 2019*, ser. LIPIcs, N. Pisanti and S. P. Pissis, Eds., vol. 128. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2019, pp. 4:1–4:12. [Online]. Available: https://doi.org/10.4230/LIPIcs.CPM.2019.4

[19] T. Gagie, P. Gawrychowski, J. Kärkkäinen, Y. Nekrich, and S. J. Puglisi, "A faster grammar-based self-index," in *6th International Conference on Language and Automata Theory and Applications, LATA 2012*, ser. LNCS, A. Dediu and C. Martín-Vide, Eds., vol. 7183. Springer, 2012, pp. 240–251. [Online]. Available: https://doi.org/10.1007/978-3-642-28332-1_21

[20] S. Kreft and G. Navarro, "On compressing and indexing repetitive sequences," *Theoretical Computer Science*, vol. 483, pp. 115–133, 2013. [Online]. Available: https://doi.org/10.1016/J.TCS.2012.02.006

[21] T. Gagie, P. Gawrychowski, J. Kärkkäinen, Y. Nekrich, and S. J. Puglisi, "LZ77-based self-indexing with faster pattern matching," in *11th Latin American Symposium on Theoretical Informatics, LATIN 2014*, ser. LNCS, A. Pardo and A. Viola, Eds., vol. 8392. Springer, 2014, pp. 731–742. [Online]. Available: https://doi.org/10.1007/978-3-642-54423-1_63

[22] H. Ferrada, T. Gagie, T. Hirvola, and S. J. Puglisi, "Hybrid indexes for repetitive datasets," *Philosophical Transactions of the Royal Society A*, vol. 372, 2014. [Online]. Available: https://doi.org/10.1098/rsta.2013.0137

[23] D. Valenzuela, "CHICO: A compressed hybrid index for repetitive collections," in *15th International Symposium on Experimental Algorithms, SEA 2016*, ser. LNCS, A. V. Goldberg and A. S. Kulikov, Eds., vol. 9685. Springer, 2016, pp. 326–338. [Online]. Available: https://doi.org/10.1007/978-3-319-38851-9_22

[24] P. Bille, M. B. Ettienne, I. L. Gørtz, and H. W. Vildhøj, "Time-space trade-offs for Lempel-Ziv compressed indexing," *Theoretical Computer Science*, vol. 713, pp. 66–77, 2018. [Online]. Available: https://doi.org/10.1016/J.TCS.2017.12.021

[25] T. Nishimoto, T. I, S. Inenaga, H. Bannai, and M. Takeda, “Dynamic index and LZ factorization in compressed space,” *Discrete Applied Mathematics*, vol. 274, pp. 116–129, 2020. [Online]. Available: https://doi.org/10.1016/J.DAM.2019.01.014

[26] A. R. Christiansen, M. B. Ettienne, T. Kociumaka, G. Navarro, and N. Prezza, “Optimal-time dictionary-compressed indexes,” *ACM Transactions on Algorithms*, vol. 17, no. 1, pp. 8:1–8:39, 2021. [Online]. Available: https://doi.org/10.1145/3426473

[27] T. Kociumaka, G. Navarro, and F. Olivares, “Near-optimal search time in δ-optimal space, and vice versa,” *Algorithmica*, vol. 13568, no. 4, pp. 1031–1056, 2022. [Online]. Available: https://doi.org/10.1007/S00453-023-01186-0

[28] D. Kempa and T. Kociumaka, “Collapsing the hierarchy of compressed data structures: Suffix arrays in optimal compressed space,” in *64th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2023*. IEEE, 2023, pp. 1877–1886. [Online]. Available: https://doi.org/10.1109/FOCS57990.2023.00114

[29] M. Charikar, E. Lehman, D. Liu, R. Panigrahy, M. Prabhakaran, A. Sahai, and A. Shelat, “The smallest grammar problem,” *IEEE Transactions on Information Theory*, vol. 51, no. 7, pp. 2554–2576, 2005. [Online]. Available: https://doi.org/10.1109/TIT.2005.850116

[30] D. Kempa and N. Prezza, “At the roots of dictionary compression: String attractors,” in *50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018*, I. Diakonikolas, D. Kempe, and M. Henzinger, Eds. ACM, 2018, pp. 827–840. [Online]. Available: https://doi.org/10.1145/3188745.3188814

[31] A. Jeż, “A really simple approximation of smallest grammar,” *Theoretical Computer Science*, vol. 616, pp. 141–150, 2016. [Online]. Available: https://doi.org/10.1016/J.TCS.2015.12.032

[32] G. Navarro, “Indexing highly repetitive string collections, part I: Repetitiveness measures,” *ACM Computing Surveys*, vol. 54, no. 2, pp. 29:1–29:31, 2021. [Online]. Available: https://doi.org/10.1145/3434399

[33] ——, “Indexing highly repetitive string collections, part II: Compressed indexes,” *ACM Computing Surveys*, vol. 54, no. 2, pp. 26:1–26:32, 2021. [Online]. Available: https://doi.org/10.1145/3432999

[34] D. Hermelin, G. M. Landau, S. Landau, and O. Weimann, “Unified compression-based acceleration of edit-distance computation,” *Algorithmica*, vol. 65, no. 2, pp. 339–353, 2013. [Online]. Available: https://doi.org/10.1007/s00453-011-9590-6

[35] A. Tiskin, “Fast distance multiplication of unit-Monge matrices,” *Algorithmica*, vol. 71, no. 4, pp. 859–888, 2015. [Online]. Available: https://doi.org/10.1007/s00453-013-9830-z

[36] P. Gawrychowski, “Faster algorithm for computing the edit distance between SLP-compressed strings,” in *19th International Symposium on String Processing and Information Retrieval, SPIRE 2012*, ser. LNCS, L. Calderón-Benavides, C. N. González-Caro, E. Chávez, and N. Ziviani, Eds., vol. 7608. Springer, 2012, pp. 229–236. [Online]. Available: https://doi.org/10.1007/978-3-642-34109-0_24

[37] A. Abboud, A. Backurs, K. Bringmann, and M. Künnemann, “Fine-grained complexity of analyzing compressed data: Quantifying improvements over decompress-and-solve,” in *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017*, C. Umans, Ed. IEEE Computer Society, 2017, pp. 192–203. [Online]. Available: https://doi.org/10.1109/FOCS.2017.26

[38] A. Ganesh, T. Kociumaka, A. Lincoln, and B. Saha, “How compression and approximation affect efficiency in string distance measures,” in *33rd Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2022*, J. S. Naor and N. Buchbinder, Eds. SIAM, 2022, pp. 2867–2919. [Online]. Available: https://doi.org/10.1137/1.9781611977073.112

[39] P. Gawrychowski, “Pattern matching in Lempel-Ziv compressed strings: Fast, simple, and deterministic,” in *19th Annual European Symposium on Algorithms, ESA 2011*, ser. LNCS, C. Demetrescu and M. M. Halldórsson, Eds., vol. 6942. Springer, 2011, pp. 421–432. [Online]. Available: https://doi.org/10.1007/978-3-642-23719-5_36

[40] A. Jeż, “Faster fully compressed pattern matching by recompression,” *ACM Transactions on Algorithms*, vol. 11, no. 3, pp. 20:1–20:43, 2015. [Online]. Available: https://doi.org/10.1145/2631920

[41] M. Ganardi and P. Gawrychowski, “Pattern matching on grammar-compressed strings in linear time,” in *33rd Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2022*, J. S. Naor and N. Buchbinder, Eds. SIAM, 2022, pp. 2833–2846. [Online]. Available: https://doi.org/10.1137/1.9781611977073.110

[42] T. Gagie, P. Gawrychowski, and S. J. Puglisi, “Approximate pattern matching in LZ77-compressed texts,” *Journal of Discrete Algorithms*, vol. 32, pp. 64–68, 2015. [Online]. Available: https://doi.org/10.1016/J.JDA.2014.10.003

[43] K. Bringmann, M. Künnemann, and P. Wellnitz, “Few matches or almost periodicity: Faster pattern matching with mismatches in compressed texts,” in *30th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019*, T. M. Chan, Ed. SIAM, 2019, pp. 1126–1145. [Online]. Available: https://doi.org/10.1137/1.9781611975482.69

[44] P. Charalampopoulos, T. Kociumaka, and P. Wellnitz, “Faster approximate pattern matching: A unified approach,” in *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020*, S. Irani, Ed. IEEE Computer Society, 2020, pp. 978–989. [Online]. Available: https://doi.org/10.1109/FOCS46700.2020.00095

[45] ——, “Faster pattern matching under edit distance : A reduction to dynamic puzzle matching and the seaweed monoid of permutation matrices,” in *63rd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2022*. IEEE, 2022, pp. 698–707. [Online]. Available: https://doi.org/10.1109/FOCS54457.2022.00072

[46] P. Ferragina, G. Manzini, T. Gagie, D. Köppl, G. Navarro, M. Striani, and F. Tosoni, “Improving matrix-vector multiplication via lossless grammar-compressed matrices,” *Proceedings of the VLDB Endowment*, vol. 15, no. 10, pp. 2175–2187, 2022. [Online]. Available: https://www.vldb.org/pvldb/vol15/p2175-tosoni.pdf

[47] T. Gagie, G. Navarro, and N. Prezza, “On the approximation ratio of Lempel-Ziv parsing,” in *13th Latin American Symposium on Theoretical Informatics, LATIN 2018*, ser. LNCS, M. A. Bender, M. Farach-Colton, and M. A. Mosteiro, Eds., vol. 10807. Springer, 2018, pp. 490–503. [Online]. Available: https://doi.org/10.1007/978-3-319-77404-6_36

[48] D. Kempa and T. Kociumaka, “Resolution of the Burrows-Wheeler Transform conjecture,” in *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020*, S. Irani, Ed. IEEE Computer Society, 2020, pp. 1002–1013. [Online]. Available: https://doi.org/10.1109/FOCS46700.2020.00097

[49] T. Kociumaka, G. Navarro, and N. Prezza, “Towards a definitive compressibility measure for repetitive sequences,” *IEEE Transactions on Information Theory*, vol. 69, no. 4, pp. 2074–2092, 2023. [Online]. Available: https://doi.org/10.1109/TIT.2022.3224382

[50] S. Kreft and G. Navarro, “LZ77-like compression with fast random access,” in *2010 Data Compression Conference, DCC 2010*. IEEE Computer Society, 2010, pp. 239–248. [Online]. Available: https://doi.org/10.1109/DCC.2010.29

[51] M. Burrows and D. J. Wheeler, “A block-sorting lossless data compression algorithm,” Digital Equipment Corporation, Palo Alto, California, Tech. Rep. 124, 1994. [Online]. Available: https://www.hpl.hp.com/techreports/Compaq-DEC/SRC-RR-124.pdf

[52] J. A. Storer and T. G. Szymanski, “Data compression via textual substitution,” *Journal of the ACM*, vol. 29, no. 4, pp. 928–951, 1982. [Online]. Available: https://doi.org/10.1145/322344.322346

[53] T. Kida, T. Matsumoto, Y. Shibata, M. Takeda, A. Shinohara, and S. Arikawa, “Collage system: A unifying framework for compressed pattern matching,” *Theoretical Computer Science*, vol. 298, no. 1, pp. 253–272, 2003. [Online]. Available: https://doi.org/10.1016/S0304-3975(02)00426-7

[54] M. Rodeh, V. R. Pratt, and S. Even, “Linear algorithm for data compression via string matching,” *Journal of the ACM*, vol. 28, no. 1, pp. 16–24, 1981. [Online]. Available: https://doi.org/10.1145/322234.322237

[55] M. Crochemore, “Transducers and repetitions,” *Theoretical Computer Science*, vol. 45, no. 1, pp. 63–86, 1986. [Online]. Available: https://doi.org/10.1016/0304-3975(86)90041-1

[56] M. G. Main, “Detecting leftmost maximal periodicities,” *Discrete Applied Mathematics*, vol. 25, no. 1-2, pp. 145–153, 1989. [Online]. Available: https://doi.org/10.1016/0166-218X(89)90051-6

[57] R. M. Kolpakov and G. Kucherov, “Finding maximal repetitions in a word in linear time,” in *40th IEEE Annual Symposium on Foundations of Computer Science, FOCS 1999*. IEEE Computer Society, 1999, pp. 596–604. [Online]. Available: https://doi.org/10.1109/SFFCS.1999.814634

[58] G. Chen, S. J. Puglisi, and W. F. Smyth, “Fast and practical algorithms for computing all the runs in a string,” in *18th Annual Symposium on Combinatorial Pattern Matching, CPM 2007*, ser. LNCS, B. Ma and K. Zhang, Eds., vol. 4580. Springer, 2007, pp. 307–315. [Online]. Available: https://doi.org/10.1007/978-3-540-73437-6_31

[59] M. Crochemore and L. Ilie, “Computing longest previous factor in linear time and applications,” *Information Processing Letters*, vol. 106, no. 2, pp. 75–80, 2008. [Online]. Available: https://doi.org/10.1016/J.IPL.2007.10.006

[60] R. M. Kolpakov and G. Kucherov, "Finding repeats with fixed gap," in *7th International Symposium on String Processing and Information Retrieval, SPIRE 2000*, P. de la Fuente, Ed. IEEE Computer Society, 2000, pp. 162–168. [Online]. Available: https://doi.org/10.1109/SPIRE.2000.878192

[61] ——, "Finding approximate repetitions under Hamming distance," *Theoretical Computer Science*, vol. 303, no. 1, pp. 135–156, 2003. [Online]. Available: https://doi.org/10.1016/S0304-3975(02)00448-6

[62] D. Gusfield and J. Stoye, "Linear time algorithms for finding and representing all the tandem repeats in a string," *Journal of Computer and System Sciences*, vol. 69, no. 4, pp. 525–546, 2004. [Online]. Available: https://doi.org/10.1016/J.JCSS.2004.03.004

[63] M. Crochemore, G. M. Landau, and M. Ziv-Ukelson, "A sub-quadratic sequence alignment algorithm for unrestricted cost matrices," in *13th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2002*, D. Eppstein, Ed. ACM/SIAM, 2002, pp. 679–688. [Online]. Available: http://dl.acm.org/citation.cfm?id=545381.545472

[64] J. Duval, R. Kolpakov, G. Kucherov, T. Lecroq, and A. Lefebvre, "Linear-time computation of local periods," *Theoretical Computer Science*, vol. 326, no. 1-3, pp. 229–240, 2004. [Online]. Available: https://doi.org/10.1016/J.TCS.2004.06.024

[65] T. Kociumaka, M. Kubica, J. Radoszewski, W. Rytter, and T. Waleń, "A linear-time algorithm for seeds computation," *ACM Transactions on Algorithms*, vol. 16, no. 2, pp. 27:1–27:23, 2020. [Online]. Available: https://doi.org/10.1145/3386369

[66] D. Gusfield, *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology*. Cambridge, UK: Cambridge University Press, 1997. [Online]. Available: https://doi.org/10.1017/cbo9780511574931

[67] M. Crochemore, L. Ilie, and W. Rytter, "Repetitions in strings: Algorithms and combinatorics," *Theoretical Computer Science*, vol. 410, no. 50, pp. 5227–5235, 2009. [Online]. Available: https://doi.org/10.1016/J.TCS.2009.08.024

[68] A. Al-Hafeedh, M. Crochemore, L. Ilie, E. Kopylova, W. F. Smyth, G. Tischler, and M. Yusufu, "A comparison of index-based Lempel-Ziv LZ77 factorization algorithms," *ACM Computing Surveys*, vol. 45, no. 1, pp. 5:1–5:17, 2012. [Online]. Available: https://doi.org/10.1145/2379776.2379781

[69] Y. Zu and B. Hua, "GLZSS: LZSS lossless data compression can be faster," in *7th Workshop on General Purpose Processing Using GPUs, GPGPU 2014*, J. Cavazos, X. Gong, and D. R. Kaeli, Eds. ACM, 2014, p. 46. [Online]. Available: https://dl.acm.org/citation.cfm?id=2576785

[70] M. Naor, "String matching with preprocessing of text and pattern," in *18th International Colloquium on Automata, Languages and Programming, ICALP 1991*, ser. LNCS, J. L. Albert, B. Monien, and M. Rodríguez-Artalejo, Eds., vol. 510. Springer, 1991, pp. 739–750. [Online]. Available: https://doi.org/10.1007/3-540-54233-7_179

[71] M. Crochemore and W. Rytter, "Efficient parallel algorithms to test square-freeness and factorize strings," *Information Processing Letters*, vol. 38, no. 2, pp. 57–60, 1991. [Online]. Available: https://doi.org/10.1016/0020-0190(91)90223-5

[72] M. Farach and S. Muthukrishnan, "Optimal parallel dictionary matching and compression (extended abstract)," in *7th Annual ACM Symposium on Parallel Algorithms and Architectures, SPAA 1995*, C. E. Leiserson, Ed. ACM, 1995, pp. 244–253. [Online]. Available: https://doi.org/10.1145/215399.215451

[73] S. T. Klein and Y. Wiseman, "Parallel Lempel Ziv coding," *Discrete Applied Mathematics*, vol. 146, no. 2, pp. 180–191, 2005. [Online]. Available: https://doi.org/10.1016/J.DAM.2004.04.013

[74] J. Shun and F. Zhao, "Practical parallel Lempel-Ziv factorization," in *2013 Data Compression Conference, DCC 2013*, A. Bilgin, M. W. Marcellin, J. Serra-Sagristà, and J. A. Storer, Eds. IEEE, 2013, pp. 123–132. [Online]. Available: https://doi.org/10.1109/DCC.2013.20

[75] L. B. Han, B. Lao, and G. Nong, "Succinct parallel Lempel-Ziv factorization on a multicore computer," *Journal of Supercomputing*, vol. 78, no. 5, pp. 7278–7303, 2022. [Online]. Available: https://doi.org/10.1007/S11227-021-04165-W

[76] A. Ozsoy and D. M. Swany, "CULZSS: LZSS lossless data compression on CUDA," in *2011 IEEE International Conference on Cluster Computing, CLUSTER 2011*. IEEE Computer Society, 2011, pp. 403–411. [Online]. Available: https://doi.org/10.1109/CLUSTER.2011.52

[77] A. Ozsoy, D. M. Swany, and A. Chauhan, "Optimizing LZSS compression on GPGPUs," *Future Generation Computer System*, vol. 30, pp. 170–178, 2014. [Online]. Available: https://doi.org/10.1016/J.FUTURE.2013.06.022

[78] J. Kärkkäinen, D. Kempa, and S. J. Puglisi, "Lempel-Ziv parsing in external memory," in *2014 Data Compression Conference, DCC 2014*, A. Bilgin, M. W. Marcellin, J. Serra-Sagristà, and J. A. Storer, Eds. IEEE, 2014, pp. 153–162. [Online]. Available: https://doi.org/10.1109/DCC.2014.78

[79] D. Kosolobov, D. Valenzuela, G. Navarro, and S. J. Puglisi, "Lempel-Ziv-like parsing in small space," *Algorithmica*, vol. 82, no. 11, pp. 3195–3215, 2020. [Online]. Available: https://doi.org/10.1007/s00453-020-00722-6

[80] D. Gibney, C. Jin, T. Kociumaka, and S. V. Thankachan, "Near-optimal quantum algorithms for bounded edit distance and Lempel-Ziv factorization," in *35th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2024*, 2024, pp. 3302–3332. [Online]. Available: https://doi.org/10.1137/1.9781611977912.118

[81] D. Kosolobov, "Lempel-Ziv factorization may be harder than computing all runs," in *32nd International Symposium on Theoretical Aspects of Computer Science, STACS 2015*, ser. LIPIcs, E. W. Mayr and N. Ollinger, Eds., vol. 30. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2015, pp. 582–593. [Online]. Available: https://doi.org/10.4230/LIPICS.STACS.2015.582

[82] J. Ellert, P. Gawrychowski, and G. Gourdel, "Optimal square detection over general alphabets," in *34th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2023*, N. Bansal and V. Nagarajan, Eds. SIAM, 2023, pp. 5220–5242. [Online]. Available: https://doi.org/10.1137/1.9781611977554.CH189

[83] T. Hagerup, "Sorting and searching on the word RAM," in *15th Annual Symposium on Theoretical Aspects of Computer Science, STACS 1998*, ser. LNCS, M. Morvan, C. Meinel, and D. Krob, Eds., vol. 1373. Springer, 1998, pp. 366–398. [Online]. Available: https://doi.org/10.1007/BFb0028575

[84] P. Weiner, "Linear pattern matching algorithms," in *14th Annual Symposium on Switching and Automata Theory, SWAT (FOCS) 1973*. IEEE Computer Society, 1973, pp. 1–11. [Online]. Available: https://doi.org/10.1109/SWAT.1973.13

[85] E. Ohlebusch and S. Gog, "Lempel-Ziv factorization revisited," in *22nd Annual Symposium on Combinatorial Pattern Matching, CPM 2011*, ser. LNCS, R. Giancarlo and G. Manzini, Eds., vol. 6661. Springer, 2011, pp. 15–26. [Online]. Available: https://doi.org/10.1007/978-3-642-21458-5_4

[86] D. Kempa and S. J. Puglisi, "Lempel-Ziv factorization: Simple, fast, practical," in *15th Meeting on Algorithm Engineering and Experiments, ALENEX 2013*, P. Sanders and N. Zeh, Eds. SIAM, 2013, pp. 103–112. [Online]. Available: https://doi.org/10.1137/1.9781611972931.9

[87] J. Kärkkäinen, D. Kempa, and S. J. Puglisi, "Linear time Lempel-Ziv factorization: Simple, fast, small," in *24th Annual Symposium on Combinatorial Pattern Matching, CPM 2013*, ser. LNCS, J. Fischer and P. Sanders, Eds., vol. 7922. Springer, 2013, pp. 189–200. [Online]. Available: https://doi.org/10.1007/978-3-642-38905-4_19

[88] K. Goto and H. Bannai, "Simpler and faster Lempel Ziv factorization," in *2013 Data Compression Conference, DCC 2013*, A. Bilgin, M. W. Marcellin, J. Serra-Sagristà, and J. A. Storer, Eds. IEEE, 2013, pp. 133–142. [Online]. Available: https://doi.org/10.1109/DCC.2013.21

[89] ——, "Space efficient linear time Lempel-Ziv factorization for small alphabets," in *2024 Data Compression Conference, DCC 2014*, A. Bilgin, M. W. Marcellin, J. Serra-Sagristà, and J. A. Storer, Eds. IEEE, 2014, pp. 163–172. [Online]. Available: https://doi.org/10.1109/DCC.2014.62

[90] J. Fischer, T. I, and D. Köppl, "Lempel Ziv computation in small space (LZ-CISS)," in *26th Annual Symposium on Combinatorial Pattern Matching, CPM 2015*, ser. LNCS, F. Cicalese, E. Porat, and U. Vaccaro, Eds., vol. 9133. Springer, 2015, pp. 172–184. [Online]. Available: https://doi.org/10.1007/978-3-319-19929-0_15

[91] W. Liu, G. Nong, W. H. Chan, and Y. Wu, "Improving a lightweight LZ77 computation algorithm for running faster," *Software: Practice and Experience*, vol. 46, no. 9, pp. 1201–1217, 2016. [Online]. Available: https://doi.org/10.1002/SPE.2377

[92] A. Hong, M. Rossi, and C. Boucher, "LZ77 via prefix-free parsing," in *25th Symposium on Algorithm Engineering and Experiments, ALENEX 2023*, G. Navarro and J. Shun, Eds. SIAM, 2023, pp. 123–134. [Online]. Available: https://doi.org/10.1137/1.9781611977561.CH11

[93] D. Okanohara and K. Sadakane, "An online algorithm for finding the longest previous factors," in *16th Annual European Symposium

on Algorithms, ESA 2008, ser. LNCS, D. Halperin and K. Mehlhorn, Eds., vol. 5193. Springer, 2008, pp. 696–707. [Online]. Available: https://doi.org/10.1007/978-3-540-87744-8_58

[94] T. Starikovskaya, "Computing Lempel-Ziv factorization online," in *37th International Symposium on Mathematical Foundations of Computer Science, MFCS 2012*, ser. LNCS, B. Rovan, V. Sassone, and P. Widmayer, Eds., vol. 7464. Springer, 2012, pp. 789–799. [Online]. Available: https://doi.org/10.1007/978-3-642-32589-2_68

[95] J. Kärkkäinen, D. Kempa, and S. J. Puglisi, "Lightweight Lempel-Ziv parsing," in *12th International Symposium on Experimental Algorithms, SEA 2013*, ser. LNCS, V. Bonifaci, C. Demetrescu, and A. Marchetti-Spaccamela, Eds., vol. 7933. Springer, 2013, pp. 139–150. [Online]. Available: https://doi.org/10.1007/978-3-642-38527-8_14

[96] J. Yamamoto, T. I, H. Bannai, S. Inenaga, and M. Takeda, "Faster compact on-line Lempel-Ziv factorization," in *31st International Symposium on Theoretical Aspects of Computer Science, STACS 2014*, ser. LIPIcs, E. W. Mayr and N. Portier, Eds., vol. 25. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2014, pp. 675–686. [Online]. Available: https://doi.org/10.4230/LIPICS.STACS.2014.675

[97] A. Policriti and N. Prezza, "Fast online Lempel-Ziv factorization in compressed space," in *22nd International Symposium on String Processing and Information Retrieval, SPIRE 2015*, ser. LNCS, C. S. Iliopoulos, S. J. Puglisi, and E. Yilmaz, Eds., vol. 9309. Springer, 2015, pp. 13–20. [Online]. Available: https://doi.org/10.1007/978-3-319-23826-5_2

[98] D. Kosolobov, "Faster lightweight Lempel-Ziv parsing," in *40th International Symposium on Mathematical Foundations of Computer Science, MFCS 2015*, ser. LNCS, G. F. Italiano, G. Pighizzini, and D. Sannella, Eds., vol. 9235. Springer, 2015, pp. 432–444. [Online]. Available: https://doi.org/10.1007/978-3-662-48054-0_36

[99] D. Belazzougui and S. J. Puglisi, "Range predecessor and Lempel-Ziv parsing," in *27th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016*, R. Krauthgamer, Ed. SIAM, 2016, pp. 2053–2071. [Online]. Available: https://doi.org/10.1137/1.9781611974331.CH143

[100] D. Köppl and K. Sadakane, "Lempel-Ziv computation in compressed space (LZ-CICS)," in *2016 Data Compression Conference, DCC 2016*, A. Bilgin, M. W. Marcellin, J. Serra-Sagristà, and J. A. Storer, Eds. IEEE, 2016, pp. 3–12. [Online]. Available: https://doi.org/10.1109/DCC.2016.38

[101] J. I. Munro, G. Navarro, and Y. Nekrich, "Space-efficient construction of compressed indexes in deterministic linear time," in *28th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017*, P. N. Klein, Ed. SIAM, 2017, pp. 408–424. [Online]. Available: https://doi.org/10.1137/1.9781611974782.26

[102] D. Kempa, "Optimal construction of compressed indexes for highly repetitive texts," in *30th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019*, T. M. Chan, Ed. SIAM, 2019, pp. 1344–1357. [Online]. Available: https://doi.org/10.1137/1.9781611975482.82

[103] J. Ellert, "Sublinear time Lempel-Ziv (LZ77) factorization," in *30th International Symposium on String Processing and Information Retrieval, SPIRE 2023*, ser. LNCS, F. M. Nardini, N. Pisanti, and R. Venturini, Eds., vol. 14240. Springer, 2023, pp. 171–187. [Online]. Available: https://doi.org/10.1007/978-3-031-43980-3_14

[104] M. Crochemore and G. Tischler, "Computing longest previous non-overlapping factors," *Information Processing Letters*, vol. 111, no. 6, pp. 291–295, 2011. [Online]. Available: https://doi.org/10.1016/J.IPL.2010.12.005

[105] M. Crochemore, L. Ilie, and W. F. Smyth, "A simple algorithm for computing the Lempel Ziv factorization," in *2008 Data Compression Conference, DCC 2008*. IEEE Computer Society, 2008, pp. 482–488. [Online]. Available: https://doi.org/10.1109/DCC.2008.36

[106] J. Fischer and V. Heun, "Space-efficient preprocessing schemes for range minimum queries on static arrays," *SIAM Journal on Computing*, vol. 40, no. 2, pp. 465–492, 2011. [Online]. Available: https://doi.org/10.1137/090779759

[107] D. Kempa and T. Kociumaka, "String synchronizing sets: Sublinear-time BWT construction and optimal LCE data structure," in *51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019*, M. Charikar and E. Cohen, Eds. ACM, 2019, pp. 756–767. [Online]. Available: https://doi.org/10.1145/3313276.3316368

[108] Y. Nekrich, "New data structures for orthogonal range reporting and range minima queries," in *32nd Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2021*, D. Marx, Ed. SIAM, 2021, pp. 1191–1205. [Online]. Available: https://doi.org/10.1137/1.9781611976465.73

[109] J. Fischer, T. Gagie, P. Gawrychowski, and T. Kociumaka, "Approximating LZ77 via small-space multiple-pattern matching," in *23rd Annual European Symposium on Algorithms, ESA 2015*, ser. LNCS, N. Bansal and I. Finocchi, Eds., vol. 9294. Springer, 2015, pp. 533–544. [Online]. Available: https://doi.org/10.1007/978-3-662-48350-3_45

[110] A. Amir, G. M. Landau, and E. Ukkonen, "Online timestamped text indexing," *Information Processing Letters*, vol. 82, no. 5, pp. 253–259, 2002. [Online]. Available: https://doi.org/10.1016/S0020-0190(01)00275-7

[111] N. J. Larsson, "Most recent match queries in on-line suffix trees," in *25th Annual Symposium on Combinatorial Pattern Matching, CPM 2014*, ser. LNCS, A. S. Kulikov, S. O. Kuznetsov, and P. A. Pevzner, Eds., vol. 8486. Springer, 2014, pp. 252–261. [Online]. Available: https://doi.org/10.1007/978-3-319-07566-2_26

[112] P. Ferragina, I. Nitto, and R. Venturini, "Bit-optimal Lempel-Ziv compression," 2008. [Online]. Available: http://arxiv.org/abs/0802.0835

[113] P. Bille, P. H. Cording, J. Fischer, and I. L. Gørtz, "Lempel-Ziv compression in a sliding window," in *28th Annual Symposium on Combinatorial Pattern Matching, CPM 2017*, ser. LIPIcs, J. Kärkkäinen, J. Radoszewski, and W. Rytter, Eds., vol. 78. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2017, pp. 15:1–15:11. [Online]. Available: https://doi.org/10.4230/LIPICS.CPM.2017.15

[114] P. Gawrychowski, M. Kosche, and F. Manea, "On the number of factors in the LZ-End factorization," in *30th International Symposium on String Processing and Information Retrieval, SPIRE 2023*, ser. LNCS, F. M. Nardini, N. Pisanti, and R. Venturini, Eds., vol. 14240. Springer, 2023, pp. 253–259. [Online]. Available: https://doi.org/10.1007/978-3-031-43980-3_20

[115] D. Kempa and D. Kosolobov, "LZ-End parsing in linear time," in *25th Annual European Symposium on Algorithms, ESA 2017*, ser. LIPIcs, K. Pruhs and C. Sohler, Eds., vol. 87. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2017, pp. 53:1–53:14. [Online]. Available: https://doi.org/10.4230/LIPICS.ESA.2017.53

[116] J. Ellert, J. Fischer, and M. R. Pedersen, "New advances in rightmost Lempel-Ziv," in *30th International Symposium on String Processing and Information Retrieval, SPIRE 2023*, ser. LNCS, F. M. Nardini, N. Pisanti, and R. Venturini, Eds., vol. 14240. Springer, 2023, pp. 188–202. [Online]. Available: https://doi.org/10.1007/978-3-031-43980-3_15

[117] D. Kempa and D. Kosolobov, "LZ-End parsing in compressed space," in *2017 Data Compression Conference, DCC 2017*, A. Bilgin, M. W. Marcellin, J. Serra-Sagristà, and J. A. Storer, Eds. IEEE, 2017, pp. 350–359. [Online]. Available: https://doi.org/10.1109/DCC.2017.73

[118] M. Babenko, P. Gawrychowski, T. Kociumaka, and T. Starikovskaya, "Wavelet trees meet suffix trees," in *26th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015*, 2015, pp. 572–591. [Online]. Available: https://doi.org/10.1137/1.9781611973730.39

[119] D. R. Clark, "Compact pat trees," Ph.D. dissertation, University of Waterloo, 1998. [Online]. Available: http://hdl.handle.net/10012/64

[120] G. Jacobson, "Space-efficient static trees and graphs," in *30th IEEE Annual Symposium on Foundations of Computer Science, FOCS 1989*, 1989, pp. 549–554. [Online]. Available: https://doi.org/10.1109/SFCS.1989.63533

[121] J. I. Munro, Y. Nekrich, and J. S. Vitter, "Fast construction of wavelet trees," *Theoretical Computer Science*, vol. 638, pp. 91–97, 2016. [Online]. Available: https://doi.org/10.1016/j.tcs.2015.11.011