2023 IEEE 64th Annual Symposium on Foundations of Computer Science (FOCS) | 979-8-3503-1894-4/23/$31.00 ©2023 IEEE | DOI: 10.1109/FOCS57990.2023.00114

Collapsing the Hierarchy of Compressed Data Structures: Suffix Arrays in Optimal Compressed Space

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*—The last two decades have witnessed a dramatic increase in the amount of highly repetitive datasets consisting of sequential data (strings, texts). Processing these massive amounts of data using conventional data structures is infeasible. This fueled the development of *compressed text indexes*, which efficiently answer various queries on a given text, typically in polylogarithmic time, while occupying space proportional to the compressed representation of the text. There exist numerous structures supporting queries ranging from simple "local" queries, such as random access, through more complex ones, including longest common extension (LCE) queries, to the most powerful queries, such as the suffix array (SA) functionality. Alongside the rich repertoire of queries followed a detailed study of the trade-off between the size and functionality of compressed indexes (see: Navarro; ACM Comput. Surv. 2021). It is widely accepted that this hierarchy of structures tells a simple story: the more powerful the queries, the more space is needed. On the one hand, random access, the most basic query, can be supported using $\mathcal{O}(\delta \log \frac{n \log \sigma}{\delta \log n})$ space (where n is the length of the text, σ is the alphabet size, and δ is the text's substring complexity), which is known to be the asymptotically smallest space sufficient to represent any string with parameters n, σ, and δ (Kociumaka, Navarro, and Prezza; IEEE Trans. Inf. Theory 2023). The other end of the hierarchy is occupied by indexes supporting the suffix array queries. The currently smallest one takes $\mathcal{O}(r \log \frac{n}{r})$ space, where $r \geq \delta$ is the number of runs in the Burrows–Wheeler Transform of the text (Gagie, Navarro, and Prezza; J. ACM 2020).**

We present a new compressed index, referred to as δ-SA, that supports the powerful SA functionality and needs only $\mathcal{O}(\delta \log \frac{n \log \sigma}{\delta \log n})$ space. This collapses the hierarchy of compressed data structures into a single point: The space *required* to represent the text is simultaneously *sufficient* to efficiently support the full SA functionality. Since suffix array queries are the most widely utilized queries in string processing and data compression, our result immediately improves the space complexity of dozens of algorithms, which can now be executed in δ-optimal compressed space. The δ-SA supports both suffix array and inverse suffix array queries in $\mathcal{O}(\log^{4+\epsilon} n)$ time (where $\epsilon > 0$ is any predefined constant).

Our second main result is an $\mathcal{O}(\delta \operatorname{polylog} n)$-time construction of the δ-SA from the Lempel–Ziv (LZ77) parsing of the text. This is the first algorithm that builds an SA index in *compressed time*, i.e., time nearly linear in the compressed input size. For highly repetitive texts, this is up to exponentially faster than the previously best algorithm, which builds an $\mathcal{O}(r \log \frac{n}{r})$-size index in $\mathcal{O}(\sqrt{\delta n} \operatorname{polylog} n)$ time.

To obtain our results, we develop numerous new techniques of independent interest. This includes deterministic restricted recompression, δ-compressed string synchronizing sets, and their construction in compressed time. We also improve many other auxiliary data structures; e.g., we show the first $\mathcal{O}(\delta \log \frac{n \log \sigma}{\delta \log n})$-size index for LCE queries along with its efficient construction from the LZ77 parsing.

***Index Terms*—data compression, text indexing, compressed indexing, suffix array**

I. Introduction

The last few decades witnessed explosive growth in the amount of data humanity generates and needs to process. Many rapidly expanding datasets consist of sequential (textual) data, such as source code in version control systems [1], results of web crawls [2], versioned document collections (such as Wikipedia) [3], and, perhaps most notably, biological sequences [4], [5]. The sizes of these datasets already reach petabytes [6] and are predicted to still get orders of magnitudes larger [7]. One of the key characteristics of this data, and what turns searching such datasets into the ultimate needle-in-a-haystack scenario, is that none of it can be discarded: in computational biology, the presence or lack of disease can depend on a single mutation [4], [7], whereas in source code repositories, a bug could be a result of a single typo.

What comes to the rescue is that these datasets are extremely redundant, e.g., genomic databases are known to be up to 99.9% repetitive [4]. Researchers have therefore turned their attention to techniques from lossless data compression. Compressing alone is not enough, however, as this renders the text unreadable. The solution lies in incorporating techniques from data compression directly into the design of *compressed algorithms* and *compressed data structures*:

- To date, compressed algorithms have been developed for numerous problems, ranging from exact [8]–[12] and approximate string matching [11], [13], [14], via computing edit distance [11], [15]–[17], to fundamental linear

This work has been partially supported by the Simons Foundation Junior Faculty Fellowship.

A full version of this paper is available at arxiv.org/abs/2308.03635.

979-8-3503-1894-4/23/$31.00 ©2023 IEEE

DOI 10.1109/FOCS57990.2023.00114

algebra operations (such as inner product, matrix-vector multiplication, and matrix multiplication) ubiquitous in machine learning [18]–[20].
- Same can be said about data structures. One can keep the data in compressed form and, at the price of a moderate (typically polylogarithmic) increase in space complexity, efficiently support various queries on the original (uncompressed) text. The currently supported queries range from the fundamental local queries like random access [21]–[23], through less local rank and select [23]–[25] or longest common extension (LCE) queries [26]–[28], to the most powerful and complex queries like pattern matching [29]–[35] and full suffix array functionality [36]. The suffix array queries that, given a rank $i \in [1\,..\,n]$, ask for the starting position of the ith lexicographically smallest suffix of the length-n text, are known to be particularly powerful, as they form the backbone of dozens of string processing and data compression algorithms [37], [38].

As the field matured, numerous ways to classify and compare different compressed structures emerged [39]–[41], resulting in hierarchies of structures ordered according to their size and functionality. As expected, structures supporting the most basic queries (such as random access) occupy the low-space regime [41], while the most powerful indexes supporting suffix array functionality, such as [36], require the most space. The natural question was thus raised: How much space is required to efficiently support each functionality? Kociumaka, Navarro, and Prezza [41] recently proved that, letting δ be the *substring complexity* of the text, n be its length, and σ be the size of the alphabet, a text can be represented in $\mathcal{O}(\delta \log \frac{n \log \sigma}{\delta \log n})$ space, and this bound is asymptotically tight as a function of δ, n, and σ. Simultaneously, they showed that it is possible to support random access and pattern-matching queries in the same space (see also [35] for improvements of pattern matching query time). Given this situation, we thus ask:

What is the space required to efficiently support the most powerful queries, such as the suffix array functionality?

a) Our Results: In this paper, we establish the following two main results:

1) We develop the first data structure, referred to as δ-SA, that uses only $\mathcal{O}(\delta \log \frac{n \log \sigma}{\delta \log n})$ space and supports efficient suffix array queries (more precisely, it answers SA and inverse SA queries in $\mathcal{O}(\log^{4+\epsilon} n)$ time, where $\epsilon > 0$ is any given constant). This collapses the existing rich hierarchy of compressed data structures (see [3], [36], [39]–[41]) into a single point: In view of our result, $\mathcal{O}(\delta \log \frac{n \log \sigma}{\delta \log n})$ is the fundamental space complexity for compressed text indexing since, on the one hand, such space is *required* to represent the string [41] (and this bound holds for all combinations of n, σ, and δ) and, on the other hand, it is already *sufficient* to support the powerful SA queries. Since the suffix array queries constitute the fundamental building block of string processing algorithms [37], [38], our result immediately implies that dozens of algorithms can be executed in this δ-*optimal* space.
2) We present an algorithm that constructs the δ-SA in $\mathcal{O}(\delta \operatorname{polylog} n)$ time from the Lempel–Ziv (LZ77) parsing of text [42]. This is the first construction of an SA index running in *compressed time*, i.e., in time nearly-linear in the compressed input size. The relevance of this result lies in the fact that LZ77 can be very efficiently approximated (using, e.g., [43]) and then converted (in compressed time) into the canonical greedy form (see Proposition II.3). At the same time, LZ77 is already strong enough to compress any string into the δ-optimal size $\mathcal{O}(\delta \log \frac{n \log \sigma}{\delta \log n})$ [41]. This makes LZ77 the perfect input in any pipeline of algorithms running in compressed time. The only similar prior construction is the $\mathcal{O}(\delta \operatorname{polylog} n)$-time construction of run-length-encoded Burrows–Wheeler Transform (BWT) from the LZ77 parsing [3]. The similarity lies in the fact that RLBWT is one of the components of the r-index of Gagie, Navarro, and Prezza [36], some versions of which are also capable of answering SA queries. Our construction, however, is much stronger than that of [3]:
   - Our algorithm builds a fully functional SA index (the δ-SA), whereas the construction from [3] builds only the run-length-encoded BWT [44], which is just a single component of the index of [36]. To date, the fastest algorithm building the complete index of [36] based on the run-length-encoded BWT required $\mathcal{O}(\sqrt{rn} \operatorname{polylog} n) = \mathcal{O}(\sqrt{\delta n} \operatorname{polylog} n)$ time [45].
   - The δ-SA uses the δ-optimal space, while the index of [36] uses more space, i.e., $\mathcal{O}(r \log \frac{n}{r})$, where $r \geq \delta$ is the number of runs in the BWT [44].
   - Our algorithm is deterministic, whereas [3] only provides a Las-Vegas randomized procedure.

On the way to our main results, we also achieve several auxiliary goals of independent interest. In particular, we describe the first data structure efficiently answering longest common extension (LCE) queries using the δ-optimal space of $\mathcal{O}(\delta \log \frac{n \log \sigma}{\delta \log n})$. Moreover, we show how to deterministically construct it from the LZ77 parsing in $\mathcal{O}(\delta \operatorname{polylog} n)$ time (Theorem III.3). We also obtain the first analogous construction of a data structure that supports random-access queries in $\mathcal{O}(\delta \log \frac{n \log \sigma}{\delta \log n})$ space (Theorem III.2); the previous such indexes [35], [41] only had $\Omega(n)$-time randomized construction algorithms.

One of the biggest technical hurdles to obtaining the above results is to simultaneously achieve

(a) δ-optimal space,
(b) polylogarithmic worst-case query time, and
(c) construction in compressed time (preferably deterministic)

for every component of the structure. Satisfying any two out of three would already constitute an improvement compared to the state-of-the-art SA indexes and their construction algorithms [3], [36]. We nevertheless show that simultaneously satisfying all three is possible.

Our main new techniques to achieve this are:

(1) deterministic restricted recompression, and
(2) δ-compressed string synchronizing sets.

Restricted recompression is a technique proposed in [46] that, as shown in [41], allows constructing an RLSLP (i.e., a run-length grammar) representing the text in δ-optimal space $\mathcal{O}(\delta \log \frac{n \log \sigma}{\delta \log n})$. The analysis in [41], however, is probabilistic, and hence yields only a Las-Vegas randomized algorithm that takes $\mathcal{O}(n)$ expected time. In this paper, we improve it not only by proposing an explicit construction, resulting in the first $\mathcal{O}(n)$-time deterministic algorithm, but we also show how to achieve $\mathcal{O}(\delta \operatorname{polylog} n)$-time construction from the LZ77 parsing. The second main new technical contribution is developing δ-compressed string synchronizing sets. String synchronizing sets [47] are a powerful symmetry-breaking mechanism with numerous applications, including algorithms for longest common substrings computation [48], indexing packed strings [47], [49], [50], dynamic suffix array [51], converting between compressed representations [3], and quantum string algorithms [52], [53]. For a given parameter $\tau \in [1 \,..\, n]$, this technique selects $\mathcal{O}(n/\tau)$ *synchronizing positions* so that positions with matching contexts are treated consistently and (except for highly periodic regions of the text) there is at least one synchronizing position among any τ consecutive positions. In [3], LZ77-compressed synchronizing sets (i.e., synchronizing sets represented by synchronizing positions located close to LZ77 phrase boundaries [42]) were used to obtain an $\mathcal{O}(\delta \operatorname{polylog} n)$-time algorithm for converting the LZ77 parsing into the run-length compressed BWT [44]. That technique, however, cannot be utilized here due to three major obstacles: First, [3] uses $\Omega(z \log n)$ space (where $z \geq \delta$ is the size of the LZ77 parsing), and hence does not meet the δ-optimal space bound requirement. Second, the algorithm in [3] is able to infer some suffix ordering only for a batch of suffixes. In other words, it is an offline solution to the problem stated in this paper. Obtaining an online solution (i.e., a data structure) requires a different approach. Finally, the synchronizing set construction used in [3] is Las-Vegas randomized and hence does not satisfy our goal of achieving deterministic construction. To overcome the first obstacle, rather than storing the synchronizing positions around the LZ77 phrase boundaries, we store them in what we call a *cover*: the set of positions in the text covered by the leftmost occurrences of substrings of some fixed length. This lets us bound the number of stored synchronizing positions in terms of the substring complexity δ. The bulk of our paper is devoted to overcoming the second obstacle. We show how to combine weighted range counting and selection queries [54] with the "range refinement" technique inspired by dynamic suffix arrays [51], which gradually shrinks the range of SA to contain only suffixes prefixed with a desired length-2^k string, to obtain the SA functionality. This requires substantial modifications compared to [51] since dynamic suffix arrays are not compressed (they use $\Theta(n \operatorname{polylog} n)$ space). Finally, to address the third challenge, we utilize a novel construction of synchronizing sets using restricted recompression, binding our last problem to the first technique. This is similar to [51], except that here we avoid the $\mathcal{O}(\log^* n)$ space increase since our structure is static. Instead, we carefully design a potential function that guides the recompression algorithm so that the outcome is deterministically as good as it would have been in expectation if we used randomization. We give a more detailed overview of our techniques in Section III. Our final result is summarized as follows.[1] Recall that $\mathrm{SA}[i]$ is the starting positions of the ith lexicographically smallest suffix of T, whereas $\mathrm{SA}^{-1}[i]$ is the lexicographic rank of $T[i \,..\, n]$ among the suffixes of T; see Section II for formal definitions.

Theorem I.1 (δ-SA). *Given the LZ77 parsing of $T \in [0 \,..\, \sigma)^n$ and any constant $\epsilon \in (0, 1)$, we can in $\mathcal{O}(\delta \log^7 n)$ time construct a data structure of size $\mathcal{O}(\delta \log \frac{n \log \sigma}{\delta \log n})$ (where δ is the substring complexity of T) that, given any position $i \in [1 \,..\, n]$, returns the values $\mathrm{SA}[i]$ and $\mathrm{SA}^{-1}[i]$ in $\mathcal{O}(\log^{4+\epsilon} n)$ time. The construction algorithm is deterministic, and the running times are worst-case.*

b) Related Work: In this paper, we focus on algorithms and data structures working for *highly repetitive* strings T, which can be defined as those for which either of the values: $z(T)$ (the size of the LZ77 parsing [42]), $\gamma^*(T)$ (the size of the smallest string attractor [39]), $g^*(T)$ (the size of the smallest context-free grammar [55]–[57]), $r(T)$ (the number of runs in the BWT [44]), or $\delta(T)$ (the substring complexity [41]) are significantly smaller than n (the list of such measures goes on [58]–[60]; see [40] for a survey). We can use either of them, since a series of papers [3], [39], [41], [56], [57], [61], [62] demonstrates that the ratio between any two of these values is $\mathcal{O}(\operatorname{polylog} n)$ for every text T of length n. The redundancy captured by these measures is present in modern massive datasets. A lot of the earlier work on small-space data structures, however, focused on reducing the sizes of structures relative to the size of text, i.e., $\mathcal{O}(n \log \sigma)$ bits (assuming $T \in [0 \,..\, \sigma)^n$) or, a step further, on achieving some variant of the kth order entropy bound $\mathcal{O}(nH_k(T)) + o(n \log \sigma)$, i.e., removing the "statistical" redundancy caused by skewed frequencies of individual symbols or short substrings of length $k = o(\log_\sigma n)$. Some of the most popular structures in this setting include those answering rank and select queries, such as wavelet trees [63], or those with pattern matching and suffix array or suffix tree functionality, such as the FM-index [64], the compressed suffix array (CSA) [65], or the compressed suffix tree (CST) [66]. Many of these structures have subsequently been implemented and are now available via libraries, such as `sdsl` [67]. The construction of these indexes has also received a lot of attention, and nowadays, most of them can be constructed very efficiently [68]–[71]. Recently, new $\mathcal{O}(n \log \sigma)$-bit indexes with CSA and CST capabilities have been proposed that also admit $o(n)$-time construction [50] if $\log \sigma = o(\sqrt{\log n})$. We refer to [72]–[75] for further details.

[1] We did not aim to optimize the $\operatorname{polylog} n$ factor in the construction algorithm. In particular, we utilized existing procedures whose running time has not been optimized either.

II. Preliminaries

a) Basic definitions: A *string* is a finite sequence of characters from a given *alphabet* Σ. The length of a string $S \in \Sigma^*$ is denoted $|S|$. For $i \in [1\,..\,|S|]$,[2] the ith character of S is denoted $S[i]$. A *substring* of S is a string of the form $S[i\,..\,j) = S[i]S[i+1]\cdots S[j-1]$ for some $1 \le i \le j \le |S|+1$. Substrings the form $S[1\,..\,j)$ and $S[i\,..\,|S|]$ 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 $S \in \Sigma^+$ is said to be *primitive* if it cannot be written as $S = U^k$, where $k \ge 2$. An integer $p \in [1\,..\,|S|]$ is a *period* of S if $S[i] = S[i+p]$ holds for every $i \in [1\,..\,|S|-p]$. We denote the shortest period of S as $\mathrm{per}(S)$. 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, $S = S^\infty[1\,..\,|S|]$. The *rotation* operation $\mathrm{rot}(\cdot)$, given a string $S \in \Sigma^+$, moves the last character of S to the front so that $\mathrm{rot}(S) = S[|S|]\cdot S[1\,..\,|S|-1]$. The inverse operation $\mathrm{rot}^{-1}(\cdot)$ moves the first character of S to the back so that $\mathrm{rot}^{-1}(S) = S[2\,..\,|S|]\cdot S[1]$. For an integer $s \in \mathbb{Z}$, the operation $\mathrm{rot}^s(\cdot)$ denotes the $|s|$-time composition of $\mathrm{rot}(\cdot)$ (if $s \ge 0$) or $\mathrm{rot}^{-1}(\cdot)$ (if $s \le 0$). Strings S, S' are *cyclically equivalent* if $S' = \mathrm{rot}^s(S)$ for some $s \in \mathbb{Z}$. By $\mathrm{lcp}(U,V)$ (resp. $\mathrm{lcs}(U,V)$) we denote the length of the longest common prefix (resp. suffix) of U and V. For any string $S \in \Sigma^*$ and any $j_1, j_2 \in [1\,..\,|S|]$, we denote $\mathrm{LCE}_S(j_1,j_2) = \mathrm{lcp}(S[j_1\,..\,|S|], S[j_2\,..\,|S|])$.

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) $U[1\,..\,i) = V[1\,..\,i)$ and $U[i] \prec V[i]$ holds for some $i \in [1\,..\,\min(|U|,|V|)]$.

Definition II.1. For any strings $T \in \Sigma^n$, $P \in \Sigma^*$, and integer $\ell \ge 0$, we define

$$\begin{aligned}\mathrm{Occ}_\ell(P,T) &= \{j' \in [1\,..\,n] : \\ &\qquad \mathrm{lcp}(P, T[j'\,..\,n]) \ge \min(|P|,\ell)\},\\ \mathrm{RangeBeg}_\ell(P,T) &= |\{j' \in [1\,..\,n] : T[j'\,..\,n] \prec P \text{ and} \\ &\qquad j' \notin \mathrm{Occ}_\ell(P,T)\}|,\\ \mathrm{RangeEnd}_\ell(P,T) &= \mathrm{RangeBeg}_\ell(P,T) + |\mathrm{Occ}_\ell(P,T)|.\end{aligned}$$

When $\ell = |P|$, we simply write $\mathrm{Occ}(P,T)$, $\mathrm{RangeBeg}(P,T)$, and $\mathrm{RangeEnd}(P,T)$, omitting the subscript. By $\mathrm{Occ}_\ell(j,T)$, $\mathrm{RangeBeg}_\ell(j,T)$, and $\mathrm{RangeEnd}_\ell(j,T)$ we also mean, respectively, $\mathrm{Occ}_\ell(T[j\,..\,n],T)$, $\mathrm{RangeBeg}_\ell(T[j\,..\,n],T)$, and $\mathrm{RangeEnd}_\ell(T[j\,..\,n],T)$.

Remark II.2. Note that the above generalizes the notation for ranges used in [50] (where the parameter ℓ was not used) and from [51] (where only patterns of the form $P = T[j\,..\,n]$ were considered). Here, we obtain both the earlier notations as special cases.

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

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

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

b) Suffix array: For any $T \in \Sigma^n$ (where $n \ge 1$), the *suffix array* $\mathrm{SA}_T[1\,..\,n]$ of T is a permutation of $[1\,..\,n]$ such that $T[\mathrm{SA}_T[1]\,..\,n] \prec T[\mathrm{SA}_T[2]\,..\,n] \prec \cdots \prec T[\mathrm{SA}_T[n]\,..\,n]$, 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* $\mathrm{ISA}_T[1\,..\,n]$ (also denoted $\mathrm{SA}_T^{-1}[1\,..\,n]$) 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 $T[j\,..\,n]$ among the suffixes of T. Note that if $T \ne \varepsilon$, then

$$\begin{aligned}&\mathrm{Occ}_\ell(P,T) = \\ &\{\mathrm{SA}_T[i] : i \in (\mathrm{RangeBeg}_\ell(P,T)\,..\,\mathrm{RangeEnd}_\ell(P,T)]\}\end{aligned}$$

holds for every $P \in \Sigma^*$ and $\ell \ge 0$. Whenever T is clear from the context, we drop the subscript in SA_T and ISA_T.

c) Substring complexity: For a string $T \in \Sigma^n$ and $\ell \in \mathbb{Z}_{>0}$, we denote the number of length-ℓ substrings by $\mathsf{d}_\ell(T) = |\{T[i\,..\,i+\ell) : i \in [1\,..\,n-\ell+1]\}|$; note that $\mathsf{d}_\ell(T) = 0$ if $\ell > n$. The *substring complexity* of T is defined as $\delta(T) = \max_{\ell=1}^{n} \frac{1}{\ell}\mathsf{d}_\ell(T)$ [41]. On the one hand, as shown in [41], the measure δ is an asymptotic lower bound for nearly all known compression algorithms and repetitiveness measures, including LZ77 [42], run-length-compressed BWT [44], grammar compression [57], and string attractors [39]. On the other hand, [41] also shows that, if $\Sigma = [0\,..\,\sigma)$, then it is possible to represent T using $\mathcal{O}(\delta(T)\log\frac{n\log\sigma}{\delta(T)\log n})$ space (or more precisely, $\mathcal{O}(\delta(T)\log\frac{n\log\sigma}{\delta(T)\log n}\log n)$ bits), and this bound is asymptotically tight as a function of n, σ, and $\delta(T)$. In other words, there is no combination of values n, σ, and δ such that every string $T \in [0\,..\,\sigma)^n$ with substring complexity $\delta(T) \le \delta$ can be encoded using $o(\delta\log\frac{n\log\sigma}{\delta\log n}\log n)$ bits.

d) Lempel–Ziv compression: A fragment $T[i\,..\,i+\ell)$ of T is a *previous factor* if it has an earlier occurrence in T, i.e., $\mathrm{LCE}_T(i,i') \ge \ell$ holds for some $i' \in [1\,..\,i)$. An *LZ77-like factorization* of T is a factorization $T = F_1\cdots F_f$ into non-empty *phrases* such that each phrase F_j with $|F_j| > 1$ is a previous factor. In the underlying *LZ77-like representation*,

every phrase $F_j = T[i\,..\,i+\ell)$ that is a previous factor is encoded as (i',ℓ), where $i' \in [1\,..\,i)$ satisfies $\mathrm{LCE}_T(i,i') \geq \ell$ (and is chosen arbitrarily in case of multiple possibilities); if $F_j = T[i]$ is not a previous factor, we encode it as $(T[i],0)$.

The LZ77 factorization [42] (or the LZ77 parsing) of a string T is then just an LZ77-like factorization constructed by greedily parsing T from left to right into the longest possible phrases. More precisely, the jth phrase F_j is the longest previous factor starting at position $1+|F_1\cdots F_{j-1}|$; if no previous factor starts there, then F_j consists of a single character. This greedy construction yields the smallest LZ77-like factorization of T [76, Theorem 1]. We denote the number of phrases in the LZ77 parsing of T by $z(T)$. For example, the text $T = \mathtt{bbabaababababaababa}$ of Fig. 1 has LZ77 factorization $\mathtt{b}\cdot\mathtt{b}\cdot\mathtt{a}\cdot\mathtt{ba}\cdot\mathtt{aba}\cdot\mathtt{bababa}\cdot\mathtt{ababa}$ with $z(T) = 7$ phrases, and its LZ77 representation is $(\mathtt{b},0),(1,1),(\mathtt{a},0),(2,2),(3,3),(7,6),(10,5)$.

Proposition II.3. *Given a string T of length n, represented using an LZ77-like parsing consisting of f phrases, the LZ77 parsing of T can be constructed in $\mathcal{O}(f\log^4 n)$ time.*

Proof. We use [77, Theorem 6.11] to build a data structure that, for any pattern P represented by its arbitrary occurrence in T, returns the leftmost occurrence of P in T. Then, we process T from left to right constructing the LZ77 parsing of T. Suppose that we have already parsed a prefix $T[1\,..\,i)$. We binary search for the maximum length ℓ such that the leftmost occurrence of $T[i\,..\,i+\ell)$ is $T[i'\,..\,i'+\ell)$ for some $i' \in [1\,..\,i)$. By definition of the LZ77 parsing, the next phrase is either $T[i]$ (if $\ell = 0$) or $T[i\,..\,i+\ell)$ (otherwise). The construction time of [77, Theorem 6.11] is $\mathcal{O}(f\log^4 n)$, whereas the query time is $\mathcal{O}(\log^3 n)$. For each phrase of the LZ77 parsing, we make $\mathcal{O}(\log n)$ queries, which take $\mathcal{O}(\log^4 n)$ time in total. Since $z(T) \leq f$, the overall running time is $\mathcal{O}(f\log^4 n)$. □

e) String Synchronizing Sets:

Definition II.4 (τ-synchronizing set [47])**.** Let $T \in \Sigma^n$ be a string and let $\tau \in [1\,..\,\lfloor\frac{n}{2}\rfloor]$ be a parameter. A set $\mathsf{S} \subseteq [1\,..\,n-2\tau+1]$ is called a *τ-synchronizing set* of T if it satisfies the following *consistency* and *density* conditions:

1) If $T[i\,..\,i+2\tau) = T[i'\,..\,i'+2\tau)$, then $i \in \mathsf{S}$ holds if and only if $i' \in \mathsf{S}$ (for $i,i' \in [1\,..\,n-2\tau+1]$),
2) $\mathsf{S} \cap [i\,..\,i+\tau) = \emptyset$ if and only if $i \in \mathsf{R}(\tau,T)$ (for $i \in [1\,..\,n-3\tau+2]$), where

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

Remark II.5. 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 $n \geq 3\tau-1$ 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 one.

Theorem II.6 ([47, Proposition 8.10], [46, Theorem 1.12])**.** *For every string T of length n and parameter $\tau \in [1\,..\,\lfloor\frac{n}{2}\rfloor]$, there exists a τ-synchronizing set S of size $|\mathsf{S}| = \mathcal{O}\left(\frac{n}{\tau}\right)$. Moreover, if $T \in [0\,..\,\sigma)^n$, where $\sigma = n^{\mathcal{O}(1)}$, then such S can be deterministically constructed in $\mathcal{O}(n)$ time.*

f) Model of computation: We use the standard word RAM model of computation [78] with w-bit *machine words*, where $w \geq \log n$, 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.

III. Technical Overview

Let $T \in \Sigma^n$, where $\Sigma = [0\,..\,\sigma)$. Assume that $T[n]$ is a symbol that does not occur in $T[1\,..\,n)$. Moreover, let $\epsilon \in (0,1)$ be a constant. In this section, we give an overview of the δ-SA, which is a compressed text index that

(a) takes $\mathcal{O}(\delta(T)\log\frac{n\log\sigma}{\delta(T)\log n})$ space,
(b) answers SA and ISA queries on T in $\mathcal{O}(\log^{4+\epsilon} n)$ time,
(c) and can be constructed from the LZ77 parsing of T in $\mathcal{O}(\delta(T)\log^7 n)$ time.

A. SA and ISA Queries

a) The Basic Idea: First, we observe that the uniqueness of $T[n]$ implies that

$$\mathrm{Occ}_\ell(j,T) = \{j' \in [1\,..\,n] : T^\infty[j'\,..\,j'+\ell) = T^\infty[j\,..\,j+\ell)\}$$

holds for every $j \in [1\,..\,n]$ and $\ell \geq 0$ (cf. Definition II.1). The main idea of the query algorithms is as follows:

- To calculate $\mathrm{ISA}[j]$ given $j \in [1\,..\,n]$, we compute the following three values for subsequent $k \in [4\,..\,\lceil\log n\rceil]$: the ranks $b = \mathrm{RangeBeg}_{2^k}(j,T)$ and $e = \mathrm{RangeEnd}_{2^k}(j,T)$ such that $\mathrm{Occ}_{2^k}(j,T) = \{\mathrm{SA}[i] : i \in (b\,..\,e]\}$ (as discussed in Section II) as well as an arbitrary position $j' \in \mathrm{Occ}_{2^k}(j,T)$ satisfying $j' = \min \mathrm{Occ}_{2^{k+1}}(j',T)$. For $k = 4$, these values are computed from scratch; subsequently, we rely on the output of the preceding step. After completing the final step, we return $\mathrm{ISA}[j] := \mathrm{RangeEnd}_\ell(j,T)$, where $\ell = 2^{\lceil\log n\rceil} \geq n$. This is the correct answer because $\mathrm{Occ}_\ell(j,T) = \mathrm{Occ}_n(j,T) = \{j\}$.
- To calculate $\mathrm{SA}[i]$ given $i \in [1\,..\,n]$, we proceed similarly, that is, we compute, for $k \in [4\,..\,\lceil\log n\rceil]$, the ranks $b = \mathrm{RangeBeg}_{2^k}(\mathrm{SA}[i],T)$ and $e = \mathrm{RangeEnd}_{2^k}(\mathrm{SA}[i],T)$ as well as an arbitrary position $j' \in \mathrm{Occ}_{2^k}(\mathrm{SA}[i],T)$ satisfying $j' = \min \mathrm{Occ}_{2^{k+1}}(j',T)$. After completing the final step, we return the position j' satisfying $j' \in \mathrm{Occ}_\ell(\mathrm{SA}[i],T)$ for $\ell = 2^{\lceil\log n\rceil} \geq n$. This is the correct answer because $\mathrm{Occ}_\ell(\mathrm{SA}[i],T) = \mathrm{Occ}_n(\mathrm{SA}[i],T) = \{\mathrm{SA}[i]\}$. Note that the individual steps of computing $\mathrm{SA}[i]$ are different from those for $\mathrm{ISA}[j]$ since we are given the rank i rather than the position $\mathrm{SA}[i]$.

This basic framework is similar to the one in [51]. The major difference, however, lies in the implementation of the "refinement" procedure: While the data structure of [51] uses $\widetilde{\Theta}(n)$ space[3], here we can only store $\widetilde{\mathcal{O}}(\delta(T))$ words. Since

[3]The $\widetilde{\mathcal{O}}(\cdot)$ notation hides factors polylogarithmic in the (uncompressed) input size n. In other words, for any function f, we have $\widetilde{\mathcal{O}}(f) = \mathcal{O}(f\,\mathrm{polylog}\,n)$. Similarly, $\widetilde{\Omega}(f) = \Omega(f/\,\mathrm{polylog}\,n)$ and $\widetilde{\Theta}(f) = \widetilde{\mathcal{O}}(f) \cap \widetilde{\Omega}(f)$.

this space allowance can be up to exponentially smaller, a much more complex approach is required.

To implement the initial step in both queries, it suffices to store all length-16 substrings of T^∞ in the lexicographic order, each augmented with the endpoints of the corresponding SA range and the position of the leftmost occurrence in $T^\infty[1..)$. Since T^∞ contains fewer than $\mathsf{d}_{16}(T)+16 \le 16\delta(T)+16 \le 32\delta(T)$ length-16 substrings, the resulting arrays need $\mathcal{O}(\delta(T))$ space. They are also easy to obtain from the LZ77 parsing: It suffices to consider all length-16 substrings overlapping phrase boundaries; for each of them, the leftmost occurrence and the number of occurrences can be determined using existing compressed text indexes [77, Theorems 6.11 and 6.21] (note that we use these indexes solely within our construction procedure; they are not included in the δ-SA). The key difficulty is thus the refinement procedure.

Definition III.1. For any $\ell \ge 1$ and $P \in \Sigma^+$, we define

$$\mathrm{Pos}^{\mathrm{beg}}_\ell(P,T) = \{j' \in \mathrm{Occ}_\ell(P,T) : T[j'..n] \prec P \text{ and } j' \notin \mathrm{Occ}_{2\ell}(P,T)\},$$
$$\mathrm{Pos}^{\mathrm{end}}_\ell(P,T) = \{j' \in \mathrm{Occ}_\ell(P,T) : T[j'..n] \succ P \text{ and } j' \notin \mathrm{Occ}_{2\ell}(P,T)\}.$$

We denote $\delta^{\mathrm{beg}}_\ell(P,T) := |\mathrm{Pos}^{\mathrm{beg}}_\ell(P,T)|$ and $\delta^{\mathrm{end}}_\ell(P,T) := |\mathrm{Pos}^{\mathrm{end}}_\ell(P,T)|$. For any position $j \in [1..n]$, we then let $\mathrm{Pos}^{\mathrm{beg}}_\ell(j,T) := \mathrm{Pos}^{\mathrm{beg}}_\ell(T[j..n],T)$ and $\mathrm{Pos}^{\mathrm{end}}_\ell(j,T) := \mathrm{Pos}^{\mathrm{end}}_\ell(T[j..n],T)$. The values $\delta^{\mathrm{beg}}_\ell(j,T)$ and $\delta^{\mathrm{end}}_\ell(j,T)$ are defined analogously.

Let us fix $k \in [4..\lceil \log n \rceil)$ and denote $\ell = 2^k$. Observe that every P satisfies

$$\mathrm{RangeBeg}_{2\ell}(P,T) = \mathrm{RangeBeg}_\ell(P,T) + \delta^{\mathrm{beg}}_\ell(P,T),$$
$$\mathrm{RangeEnd}_{2\ell}(P,T) = \mathrm{RangeEnd}_\ell(P,T) - \delta^{\mathrm{end}}_\ell(P,T).$$

In particular, every position $j \in [1..n]$ satisfies $\mathrm{RangeBeg}_{2\ell}(j,T) = \mathrm{RangeBeg}_\ell(j,T) + \delta^{\mathrm{beg}}_\ell(j,T)$ and $\mathrm{RangeEnd}_{2\ell}(j,T) = \mathrm{RangeEnd}_\ell(j,T) - \delta^{\mathrm{end}}_\ell(j,T)$. Thus, to refine the suffix array range, it suffices to compute any two values among $\delta^{\mathrm{beg}}_\ell(j,T), \delta^{\mathrm{end}}_\ell(j,T), |\mathrm{Occ}_{2\ell}(j,T)|$ (during an ISA query) or among $\delta^{\mathrm{beg}}_\ell(\mathrm{SA}[i],T), \delta^{\mathrm{end}}_\ell(\mathrm{SA}[i],T), |\mathrm{Occ}_{2\ell}(\mathrm{SA}[i],T)|$ (during an SA query).

Denote $\tau = \lfloor \frac{\ell}{3} \rfloor$ and let R be a shorthand for

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

The refinement step during the computation of $\mathrm{ISA}[j]$ (resp. $\mathrm{SA}[i]$) works differently depending on whether $j \in \mathsf{R}$ (resp. $\mathrm{SA}[i] \in \mathsf{R}$), in which case we call j (resp. $\mathrm{SA}[i]$) *periodic*. Otherwise, the position is called *nonperiodic*. To distinguish these two cases, we store the set $\mathsf{R} \cap \mathsf{C}$, where C is a 14τ-cover of T, i.e., a subset of $[1..n]$ including all positions covered by the leftmost occurrences of length-14τ substrings of T. As $\mathsf{R} \cap \mathsf{C}$ might be large, we store its *interval representation* $\mathcal{I}(\mathsf{R} \cap \mathsf{C})$, that is, we express $\mathsf{R} \cap \mathsf{C}$ as a union of disjoint integer intervals.

Observation 1: There exists a 14τ-cover C such that $|\mathcal{I}(\mathsf{C})| = \mathcal{O}(\delta(T))$ and $\mathcal{I}(\mathsf{C})$ admits a fast construction algorithm from the LZ77 parsing of T. Let C be the union of $(n-28\tau..n]$ as well as intervals $[x..x+28\tau)$ over all $x \in [1..n-28\tau]$ such that $x \equiv 1 \pmod{14\tau}$ and $x = \min \mathrm{Occ}_{28\tau}(x,T)$. The set C is a 14τ-cover since the leftmost occurrence of every length-14τ substring of T can be extended into an interval $[x..x+28\tau)$ for x as above. Note that C is a subset of positions covered by the leftmost occurrences of *all* length-28τ substring of T. By [41], we thus have $|\mathsf{C}| \le 84\tau\delta(T)$. On the other hand, C is a union of length-28τ intervals. Thus, $|\mathcal{I}(\mathsf{C})| = \mathcal{O}(|\mathsf{C}|/\tau) = \mathcal{O}(\delta(T))$. To construct $\mathcal{I}(\mathsf{C})$, it suffices to check at most two positions around each LZ77 phrase boundary. Using an index for finding leftmost occurrences [77, Theorem 6.11], we can thus build $\mathcal{I}(\mathsf{C})$ in $\mathcal{O}(z(T)\,\mathrm{polylog}\,n) = \mathcal{O}(\delta(T)\,\mathrm{polylog}\,n)$ time.

Observation 2: The above C satisfies $|\mathcal{I}(\mathsf{R}\cap\mathsf{C})| = \mathcal{O}(\delta(T))$ and $\mathcal{I}(\mathsf{R}\cap\mathsf{C})$ also admits fast construction. First, we observe that any two maximal blocks of consecutive positions in R are separated by a gap of size $\Omega(\tau)$. This implies that, using the above C, the interval representation of $\mathsf{C}\cap\mathsf{R}$ is of size $\mathcal{O}(\delta(T))$. Above, we noted that $\mathcal{I}(\mathsf{C})$ can be constructed from the LZ77 parsing in $\mathcal{O}(\delta(T)\,\mathrm{polylog}\,n)$ time. It remains to observe that, given $\mathcal{I}(\mathsf{C})$, constructing $\mathcal{I}(\mathsf{R}\cap\mathsf{C})$ reduces to computing the shortest periods via so-called 2-period queries. Consequently, by utilizing an existing index for 2-period queries [77, Theorem 6.7], we can construct $\mathcal{I}(\mathsf{R}\cap\mathsf{C})$ in $\mathcal{O}(\delta(T)\,\mathrm{polylog}\,n)$ time.

Observation 3: Using $\mathcal{I}(\mathsf{R}\cap\mathsf{C})$, we can efficiently check if $j \in \mathsf{R}$ (resp. $\mathrm{SA}[i] \in \mathsf{R}$). Recall that, at the beginning of the refinement step, we have some $j' \in \mathrm{Occ}_\ell(j,T)$ (resp. $j' \in \mathrm{Occ}_\ell(\mathrm{SA}[i],T)$). By $3\tau - 1 \le \ell$ and the definition of R, we thus have $j \in \mathsf{R}$ (resp. $\mathrm{SA}[i] \in \mathsf{R}$) if and only if $j' \in \mathsf{R}$. Moreover, since j' satisfies $j' = \min \mathrm{Occ}_{2\ell}(j',T)$, and any 14τ-cover is also a 2ℓ-cover, it thus follows that $j' \in \mathsf{C}$. Thus, $j' \in \mathsf{C}\cap\mathsf{R}$ if and only if $j' \in \mathsf{R}$. Consequently, to check if $j' \in \mathsf{R}$, it suffices to test whether j' belongs to any interval contained in $\mathcal{I}(\mathsf{R}\cap\mathsf{C})$. Provided that the intervals in $\mathcal{I}(\mathsf{R}\cap\mathsf{C})$ are ordered left-to-right, this takes $\mathcal{O}(\log|\mathcal{I}(\mathsf{R}\cap\mathsf{C})|) = \mathcal{O}(\log n)$ time.

The above is a simplified analysis, and reaching $\mathcal{O}(\delta(T))$ space for each $k \in [4..\lceil\log n\rceil)$ is still not enough to achieve the δ-optimal bound of $\mathcal{O}(\delta(T)\log\frac{n\log\sigma}{\delta(T)\log n})$ for the entire structure. In our complete analysis, we prove a tighter upper bound: $|\mathcal{I}(\mathsf{R}\cap\mathsf{C})| = \mathcal{O}(\frac{1}{\ell}\mathsf{d}_{38\ell}(T)+1)$.

b) The Nonperiodic Positions: Assume $j \in [1..n]\setminus\mathsf{R}$ (resp. $\mathrm{SA}[i] \in [1..n]\setminus\mathsf{R}$). We first focus on computing $\mathrm{ISA}[j]$. Recall that we are given $b = \mathrm{RangeBeg}_\ell(j,T)$, $e = \mathrm{RangeEnd}_\ell(j,T)$, and some $j' \in \mathrm{Occ}_\ell(j,T)$ satisfying $j' = \min \mathrm{Occ}_{2\ell}(j',T)$ as input. The refinement step for nonperiodic positions first computes the position $j'' = \min \mathrm{Occ}_{2\ell}(j,T)$ (this condition implies $j'' = \min \mathrm{Occ}_{4\ell}(j'',T)$), and then the values $\delta^{\mathrm{beg}}_\ell(j,T)$ and $|\mathrm{Occ}_{2\ell}(j,T)|$. By the discussion follow-

ing Definition III.1, this is sufficient to infer $\mathrm{RangeBeg}_{2\ell}(j,T)$ and $\mathrm{RangeEnd}_{2\ell}(j,T)$.

Let S be a τ-synchronizing set of T (Definition II.4). Observe that, by $3\tau \le \ell < n$, the uniqueness of $T[n]$ in T yields $\mathrm{per}(T[n-3\tau+2\,..\,n]) > \frac{1}{3}\tau$. Thus, $n-3\tau+2 \notin \mathsf{R}$, and hence the density condition (Definition II.4(2)) implies $\mathsf{S} \cap [n-3\tau+2\,..\,n-2\tau+2) \neq \emptyset$. Consequently, $\max \mathsf{S} \ge n-3\tau+2$, and, for every $p \in [1\,..\,n-3\tau+2]$, we can define $\mathrm{succ}_{\mathsf{S}}(p) = \min\{s \in \mathsf{S} : s \ge p\}$. If $j > n-3\tau-2$, the uniqueness of $T[n]$ in T implies $\mathrm{Occ}_\ell(j,T) = \{j\}$. Thus, we henceforth assume $j \in [1\,..\,n-3\tau+2] \setminus \mathsf{R}$.

Observation 1: The set $\mathrm{Occ}_{2\ell}(j,T)$ can be characterized using S and range queries. First, note that the assumption $j \in [1\,..\,n-3\tau+2] \setminus \mathsf{R}$ and the density condition (Definition II.4(2)) yield $\mathrm{succ}_{\mathsf{S}}(j) - j < \tau$. On the other hand, by $3\tau - 1 \le 2\ell$ and the consistency condition (Definition II.4(1)), every $p \in \mathrm{Occ}_{2\ell}(j,T)$ satisfies $\mathrm{succ}_{\mathsf{S}}(p) - p = \mathrm{succ}_{\mathsf{S}}(j) - j$. Thus, letting $x_1 = \overline{T^\infty[j\,..\,\mathrm{succ}_{\mathsf{S}}(j))}$, and $y_1 = T^\infty[\mathrm{succ}_{\mathsf{S}}(j)\,..\,j+2\ell)$, the set $\mathrm{Occ}_{2\ell}(j,T)$ consists of all positions of the form $s-|x_1|$, where $s \in \mathsf{S}$ and s is preceded by $\overline{x_1}$ and followed by y_1 in T^∞. In other words,

$$\mathrm{Occ}_{2\ell}(j,T) = \{s-|x_1| : s \in \mathsf{S} \text{ and } T^\infty[s-|x_1|\,..\,s+|y_1|) = \overline{x_1}\cdot y_1\}.$$

If we further denote $c = \max\Sigma$, $x_2 = x_1 c^\infty$, and $y_2 = y_1 c^\infty$, we have

$$\mathrm{Occ}_{2\ell}(j,T) = \{s-|x_1| : s \in \mathsf{S},\ x_1 \preceq \overline{T^\infty[s-7\tau\,..\,s)} \prec x_2,\ \text{and}\ y_1 \preceq T^\infty[s\,..\,s+7\tau) \prec y_2\}$$

due to $|x_1|, |y_1| \le 7\tau$. Thus, letting

$$\mathcal{P} = \{(\overline{T^\infty[s-7\tau\,..\,s)}, T^\infty[s\,..\,s+7\tau), s) : s \in \mathsf{S}\}$$

be a set of labeled points, the set $\mathrm{Occ}_{2\ell}(j,T)$ shifted by $|x_1|$ forward consists of the labels of the points in the range $[x_1\,..\,x_2) \times [y_1\,..\,y_2)$.

By the above, computing $\min \mathrm{Occ}_{2\ell}(j,T)$ reduces to an orthogonal range minimum query, returning the minimum label of a point occurring in the rectangle $[x_1\,..\,x_2) \times [y_1\,..\,y_2)$. Implementing such queries, however, is challenging. First, the coordinates of points in $\mathcal{P}$ are substrings of T^∞. Comparing them reduces to LCE and random access queries, and hence it suffices to only store labels of points in $\mathcal{P}$, i.e., the set S. The smallest prior structure for LCE queries [27], however, does not match our space bound. To avoid this issue, we develop the first structure that uses δ-optimal space $\mathcal{O}(\delta(T)\log\frac{n\log\sigma}{\delta(T)\log n})$. In addition, we describe its $\mathcal{O}(\delta(T)\,\mathrm{polylog}\,n)$-time deterministic construction from the LZ77 parsing [42] (see Theorem III.3). We also describe the first deterministic construction of the δ-optimal-space structure for random access queries (Theorem III.2). We elaborate more on these indexes in Section III-B.

The challenge thus reduces to storing and querying S. The plain representation is too large since it is not possible to reduce $|\mathsf{S}|$ below $\Theta(\frac{n}{\tau})$ in the worst case (see Remark II.5). We thus need to store S in a compressed form. Our starting point is the technique introduced in [3], which compresses S by only keeping elements of S that are within distance $\Theta(\tau)$ from LZ77 phrase boundaries. These *LZ77-compressed τ-synchronizing sets*, however, do not meet the δ-optimal space bound and come only with Las-Vegas randomized construction [3]. To solve the space issue, we again employ a 14τ-cover C of T. Denote $\mathsf{S}_{\mathrm{comp}} = \mathsf{S} \cap \mathsf{C}$ and

$$\mathcal{P}_{\mathrm{comp}} = \{(\overline{T^\infty[s-7\tau\,..\,s)}, T^\infty[s\,..\,s+7\tau), s) : s \in \mathsf{S}_{\mathrm{comp}}\}.$$

Observation 2: We can compute x_1, x_2, y_1, and y_2 using $\mathsf{S}_{\mathrm{comp}}$. Observe that, since $|x_1| + |y_1| = 2\ell$ and all strings in question occur near j, the difficulty lies in computing $|x_1|$. Recall that we are given some $j' \in \mathrm{Occ}_\ell(j,T)$ satisfying $j' = \min \mathrm{Occ}_{2\ell}(j',T)$ as input. By $3\tau - 1 \le \ell$, the consistency of S (Definition II.4(1)) yields $|x_1| = \mathrm{succ}_{\mathsf{S}}(j) - j = \mathrm{succ}_{\mathsf{S}}(j') - j'$. On the other hand, since $j' = \min \mathrm{Occ}_{2\ell}(j',T)$ and C is also a 2ℓ-cover, it follows that $[j'\,..\,j'+2\ell) \cap [1\,..\,n] \subseteq \mathsf{C}$. Thus $\mathrm{succ}_{\mathsf{S}}(j) - j < \tau$ implies $\mathrm{succ}_{\mathsf{S}}(j') \in \mathsf{S}_{\mathrm{comp}}$. Consequently, it suffices to store the sorted set $\mathsf{S}_{\mathrm{comp}}$. Given j', we can then quickly determine

$$\mathrm{succ}_{\mathsf{S}_{\mathrm{comp}}}(j') - j' = \mathrm{succ}_{\mathsf{S}}(j') - j' = \mathrm{succ}_{\mathsf{S}}(j) - j = |x_1|.$$

Observation 3: Orthogonal range minimum queries on $\mathcal{P}_{\mathrm{comp}}$ and $\mathcal{P}$ are equivalent. Let $x_1, x_2, y_1, y_2 \in \Sigma^*$ and let s_{comp} (resp. s) denote the output of the range minimum query in $[x_1\,..\,x_2)\times[y_1\,..\,y_2)$ on $\mathcal{P}_{\mathrm{comp}}$ (resp. $\mathcal{P}$). Observe that $\mathsf{S}_{\mathrm{comp}} \subseteq \mathsf{S}$ implies $s \le s_{\mathrm{comp}}$. To show the opposite inequality, define $s_{\min} \le s$ so that

$$s_{\min} = \min\{i \in [1\,..\,n] : T^\infty[i-7\tau\,..\,i+7\tau) = T^\infty[s-7\tau\,..\,s+7\tau)\}.$$

Then, either $s_{\min} \in [1\,..\,7\tau] \cup (n-7\tau\,..\,n]$ or $s_{\min} - 7\tau = \min \mathrm{Occ}_{14\tau}(s_{\min}-7\tau, T)$. In both cases, $s_{\min} \in \mathsf{C}$. Moreover, by the consistency of S (Definition II.4(1)), $s_{\min} \in \mathsf{S}$. Thus, $s_{\min} \in \mathsf{S}_{\mathrm{comp}}$. Finally, $T^\infty[s_{\min}-7\tau\,..\,s_{\min}+7\tau) = T^\infty[s-7\tau\,..\,s+7\tau)$ implies that $(\overline{T^\infty[s_{\min}-7\tau\,..\,s_{\min})}, T^\infty[s_{\min}\,..\,s_{\min}+7\tau)) \in [x_1\,..\,x_2)\times[y_1\,..\,y_2)$ holds if and only if $(\overline{T^\infty[s-7\tau\,..\,s)}, T^\infty[s\,..\,s+7\tau)) \in [x_1\,..\,x_2)\times[y_1\,..\,y_2)$. Thus, $s_{\mathrm{comp}} \le s_{\min} \le s$.

By the above, it suffices to use $\mathcal{P}_{\mathrm{comp}}$ during the computation of $\min \mathrm{Occ}_{2\ell}(j,T)$. The computation of $\delta_\ell^{\mathrm{beg}}(j,T)$ and $|\mathrm{Occ}_{2\ell}(j,T)|$ uses similar ideas, except range minimum queries are replaced with range counting. For this, we augment each point with a *weight* storing the frequency of the corresponding substring. The correctness of this follows by the local consistency of S. To avoid double counting, we also need to ensure that no two points in $\mathcal{P}_{\mathrm{comp}}$ coincide. To simultaneously still allow range minimum queries, we thus leave only points with the smallest labels.

Let us now return to computing $\mathrm{SA}[i]$. Observe that, to compute $\min \mathrm{Occ}_{2\ell}(j,T)$, $\delta_\ell^{\mathrm{beg}}(j,T)$, and $|\mathrm{Occ}_{2\ell}(j,T)|$, we

needed some occurrences of strings x_1 and y_1 satisfying $|x_1| = \mathrm{succ}_{\mathsf{S}}(j) - j$ and $T^{\infty}[j\,..\,j+2\ell) = \overline{x_1}y_1$. The input of the refinement procedure in the computation of $\mathrm{SA}[i]$, however, does not include any element of $\mathrm{Occ}_{2\ell}(\mathrm{SA}[i], T)$. Consequently, our query procedure performs an additional step that retrieves some position $p \in \mathrm{Occ}_{2\ell}(\mathrm{SA}[i], T)$. Such a position can be obtained from $\mathcal{P}_{\mathrm{comp}}$ using the inverse of range counting, i.e., range selection queries. The rest of the query execution proceeds similarly to the computation of $\mathrm{ISA}[j]$ for $j = \mathrm{SA}[i]$, except we can now use p instead of j since $p \in \mathrm{Occ}_{2\ell}(\mathrm{SA}[i], T)$ implies that $\mathrm{Occ}_{2\ell}(\mathrm{SA}[i], T) = \mathrm{Occ}_{2\ell}(p, T)$.

The remaining challenge is ensuring that $\mathsf{S}_{\mathrm{comp}}$ is small. Unlike for the compressed version of R, where upper bounds on $|\mathsf{C}|$ and $|\mathcal{I}(\mathsf{C})|$ already impose an upper bound on $|\mathcal{I}(\mathsf{R} \cap \mathsf{C})|$, the size of $\mathsf{S} \cap \mathsf{C}$ can be large if S is not constructed carefully. In this paper, we develop a deterministic construction that ensures that the total size of $\mathsf{S}_{\mathrm{comp}}$ across all levels $k \in [4\,..\,\lceil \log n \rceil)$ of the data structure is $\mathcal{O}(\delta(T) \log \frac{n \log \sigma}{\delta(T) \log n})$ (see Section III-B).

c) The Periodic Positions: Let us now assume $j \in \mathsf{R}$ (resp. $\mathrm{SA}[i] \in \mathsf{R}$). Compared to previous work using the "refinement" framework [51], one of the key challenges is as follows. The basic property of every $p \in \mathsf{R}$ (extending easily to blocks of such positions), dictating the rest of the query algorithm, is its *type*, defined as either -1 or $+1$ depending on whether the symbol following the periodic substring is larger or smaller than the symbol that would extend the period. Dealing with positions of each type is straightforward if $\tilde{\Theta}(n)$ space is available: We separately store all maximal blocks of positions in R of each type [51]. In compressed space, however, we are much more constrained. For example, $j \in \mathsf{R}$ and $j' \in \mathrm{Occ}_{2\ell}(j, T)$ may have different types, so we cannot distinguish the type simply based on the occurrence of a periodic fragment. This requires numerous new and more general combinatorial properties, allowing separate processing of elements of $\mathrm{Occ}_{2\ell}(j, T)$ (resp. $\mathrm{Occ}_{2\ell}(\mathrm{SA}[i], T)$) depending on whether they are *partially periodic* (meaning that the length of their periodic prefix is less than 2ℓ) or *fully periodic* (otherwise).

B. Deterministic Restricted Recompression

Restricted recompression [46] is a general technique for constructing a run-length grammar (RLSLP) of a given text. Utilizing this technique, Kociumaka, Navarro, and Prezza [41] proved that every text $T \in [0\,..\,\sigma)^n$ can be represented using an RLSLP of size $\mathcal{O}(\delta(T) \log \frac{n \log \sigma}{\delta(T) \log n})$ and height $\mathcal{O}(\log n)$. They also showed that $\mathcal{O}(\delta(T) \log \frac{n \log \sigma}{\delta(T) \log n})$ is the asymptotically optimal space to represent a string, for all combinations of n, σ, and $\delta(T)$. Consequently, random access to T can be efficiently supported in the δ-optimal space. Finally, they developed an $\mathcal{O}(n)$-expected-time Las-Vegas randomized construction of such RLSLP. At the heart of their construction is the problem of approximating the directed max-cut of graphs derived from partially compressed representations of T. In [41], it is proved that a uniformly random partition at every level of the grammar is sufficient to achieve the δ-optimal total size in expectation. In this paper, we describe an explicit partitioning technique resulting in the same bound on the size of the RLSLP. The unique component of our construction is the use of a *cover hierarchy*, allowing us to account for the effects of partitioning at the current level of the grammar on the properties of the grammar at all future levels. In addition, we develop an $\mathcal{O}(\delta(T) \operatorname{polylog} n)$-time deterministic construction algorithm of our RLSLP from the LZ77 parsing of T [42].

Equipped with this RLSLP, one can answer random access and LCE queries on T in $\mathcal{O}(\log n)$ time.

Theorem III.2. *For every text $T \in [0\,..\,\sigma)^n$, there exists a data structure of size $\mathcal{O}(\delta(T) \log \frac{n \log \sigma}{\delta(T) \log n})$ that given any position $i \in [1\,..\,n]$, returns $T[i]$ in $\mathcal{O}(\log n)$ time. Moreover, it can be constructed in $\mathcal{O}(\delta(T) \log^7 n)$ time given the LZ77-parsing of T.*

Theorem III.3. *For every text $T \in [0\,..\,\sigma)^n$, there exists a data structure of size $\mathcal{O}(\delta(T) \log \frac{n \log \sigma}{\delta(T) \log n})$ answering LCE_T and $\mathrm{LCE}_{\overline{T}}$ queries in $\mathcal{O}(\log n)$ time. Moreover, it can be constructed in $\mathcal{O}(\delta(T) \log^7 n)$ time given the LZ77-parsing of T.*

Furthermore, in $\mathcal{O}(\delta(T) \operatorname{polylog} n)$ time, we can derive from the RLSLP a sequence of string synchronizing sets such that, after pairwise intersection with the cover hierarchy guiding the RLSLP construction, we obtain their representation of total size $\mathcal{O}(\delta(T) \log \frac{n \log \sigma}{\delta(T) \log n})$.

References

[1] G. Navarro, "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

[2] P. Ferragina and G. Manzini, "On compressing the textual web," in *3rd International Conference on Web Search and Web Data Mining, WSDM 2010*, B. D. Davison, T. Suel, N. Craswell, and B. Liu, Eds. ACM, 2010, pp. 391–400. [Online]. Available: https://doi.org/10.1145/1718487.1718536

[3] 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

[4] M. Przeworski, R. R. Hudson, and A. D. Rienzo, "Adjusting the focus on human variation," *Trends in Genetics*, vol. 16, no. 7, pp. 296–302, 2000. [Online]. Available: https://doi.org/10.1016/S0168-9525(00)02030-8

[5] B. Berger, N. M. Daniels, and Y. W. Yu, "Computational biology in the 21st century: Scaling with compressive algorithms," *Communications of the ACM*, vol. 59, no. 8, pp. 72–80, 2016. [Online]. Available: https://doi.org/10.1145/2957324

[6] M. Hernaez, D. Pavlichin, T. Weissman, and I. Ochoa, "Genomic data compression," *Annual Review of Biomedical Data Science*, vol. 2, no. 1, pp. 19–37, 2019. [Online]. Available: https://doi.org/10.1146/annurev-biodatasci-072018-021229

[7] National Human Genome Research Institute (NIH), "Genomic data science," 2022, accessed March 30, 2023. [Online]. Available: https://www.genome.gov/about-genomics/fact-sheets/Genomic-Data-Science

[8] 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

[9] ——, "Optimal pattern matching in LZW compressed strings," *ACM Transactions on Algorithms*, vol. 9, no. 3, pp. 25:1–25:17, 2013. [Online]. Available: https://doi.org/10.1145/2483699.2483705

[10] 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

[11] 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

[12] 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

[13] 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

[14] 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

[15] 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

[16] 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

[17] 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

[18] A. Abboud, A. Backurs, K. Bringmann, and M. Künnemann, "Impossibility results for grammar-compressed linear algebra," in *34th Conference on Neural Information Processing System, NeurIPS 2020*, H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, Eds., vol. 33. Curran Associates, Inc., 2020, pp. 8810–8823. [Online]. Available: https://proceedings.neurips.cc/paper/2020/file/645e6bfdd05d1a69c5e47b20f0a91d46-Paper.pdf

[19] 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

[20] A. P. Francisco, T. Gagie, D. Köppl, S. Ladra, and G. Navarro, "Graph compression for adjacency-matrix multiplication," *SN Computer Science*, vol. 3, no. 3, p. 193, 2022. [Online]. Available: https://doi.org/10.1007/s42979-022-01084-2

[21] 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

[22] 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

[23] 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

[24] 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

[25] 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

[26] 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

[27] 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

[28] 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

[29] F. Claude and G. Navarro, "Self-indexed grammar-based compression," *Fundamenta Informaticae*, vol. 111, no. 3, pp. 313–337, 2011. [Online]. Available: https://doi.org/10.3233/FI-2011-565

[30] 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

[31] ——, "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

[32] F. Claude, G. Navarro, and A. Pacheco, "Grammar-compressed indexes with logarithmic search time," *Journal of Computer and System Sciences*, vol. 118, pp. 53–74, 2021. [Online]. Available: https://doi.org/10.1016/j.jcss.2020.12.001

[33] D. Díaz-Domínguez, G. Navarro, and A. Pacheco, "An LMS-based grammar self-index with local consistency properties," in *28th International Symposium on String Processing and Information Retrieval SPIRE 2021*, ser. LNCS, T. Lecroq and H. Touzet, Eds., vol. 12944. Springer, 2021, pp. 100–113. [Online]. Available: https://doi.org/10.1007/978-3-030-86692-1_9

[34] 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

[35] T. Kociumaka, G. Navarro, and F. Olivares, "Near-optimal search time in δ-optimal space," in *15th Latin American Symposium on Theoretical Informatics, LATIN 2022*, ser. LNCS, A. Castañeda and F. Rodríguez-Henríquez, Eds., vol. 13568. Springer, 2022, pp. 88–103. [Online]. Available: https://doi.org/10.1007/978-3-031-20624-5_6

[36] T. Gagie, G. Navarro, and N. Prezza, "Fully functional suffix trees and optimal text searching in BWT-runs bounded space," *Journal of the ACM*, vol. 67, no. 1, pp. 2:1–2:54, 2020. [Online]. Available: https://doi.org/10.1145/3375890

[37] 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

[38] D. Adjeroh, T. Bell, and A. Mukherjee, *The Burrows-Wheeler Transform: Data Compression, Suffix Arrays, and Pattern Matching*. Boston, MA, USA: Springer, 2008. [Online]. Available: https://doi.org/10.1007/978-0-387-78909-5

[39] 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

[40] 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

[41] 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

[42] 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

[43] 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

[44] 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

[45] D. Gibney and S. V. Thankachan, "Compressibility-aware quantum algorithms on strings," 2023. [Online]. Available: https://arxiv.org/abs/2302.07235

[46] T. Kociumaka, J. Radoszewski, W. Rytter, and T. Waleń, "Internal pattern matching queries in a text and applications," 2023. [Online]. Available: https://arxiv.org/abs/1311.6235v5

[47] 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

[48] P. Charalampopoulos, T. Kociumaka, S. P. Pissis, and J. Radoszewski, "Faster algorithms for longest common substring," in *29th Annual European Symposium on Algorithms, ESA 2021*, ser. LIPIcs, P. Mutzel, R. Pagh, and G. Herman, Eds., vol. 204. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2021, pp. 30:1–30:17. [Online]. Available: https://doi.org/10.4230/LIPIcs.ESA.2021.30

[49] P. Dinklage, J. Fischer, A. Herlez, T. Kociumaka, and F. Kurpicz, "Practical performance of space efficient data structures for longest common extensions," in *28th Annual European Symposium on Algorithms, ESA 2020*, ser. LIPIcs, F. Grandoni, G. Herman, and P. Sanders, Eds., vol. 173. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2020, pp. 39:1–39:20. [Online]. Available: https://doi.org/10.4230/LIPIcs.ESA.2020.39

[50] D. Kempa and T. Kociumaka, "Breaking the $O(n)$-barrier in the construction of compressed suffix arrays and suffix trees," in *34th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2023*, N. Bansal and V. Nagarajan, Eds. SIAM, 2023, pp. 5122–5202. [Online]. Available: https://doi.org/10.1137/1.9781611977554.ch187

[51] ——, "Dynamic suffix array with polylogarithmic queries and updates," in *54th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2022*, S. Leonardi and A. Gupta, Eds. ACM, 2022, pp. 1657–1670. [Online]. Available: https://doi.org/10.1145/3519935.3520061

[52] S. Akmal and C. Jin, "Near-optimal quantum algorithms for string problems," in *33rd Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2022*, J. S. Naor and N. Buchbinder, Eds. SIAM, 2022, pp. 2791–2832. [Online]. Available: https://doi.org/10.1137/1.9781611977073.109

[53] C. Jin and J. Nogler, "Quantum speed-ups for string synchronizing sets, longest common substring, and k-mismatch matching," in *34th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2023*, N. Bansal and V. Nagarajan, Eds. SIAM, 2023, pp. 5090–5121. [Online]. Available: https://doi.org/10.1137/1.9781611977554.ch186

[54] B. Chazelle, "A functional approach to data structures and its use in multidimensional searching," *SIAM Journal on Computing*, vol. 17, no. 3, pp. 427–462, 1988. [Online]. Available: https://doi.org/10.1137/0217026

[55] J. C. Kieffer and E. Yang, "Grammar-based codes: A new class of universal lossless source codes," *IEEE Transactions on Information Theory*, vol. 46, no. 3, pp. 737–754, 2000. [Online]. Available: https://doi.org/10.1109/18.841160

[56] 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

[57] 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

[58] 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

[59] 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

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

[61] 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

[62] 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

[63] R. Grossi, A. Gupta, and J. S. Vitter, "High-order entropy-compressed text indexes," in *14th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2003*. ACM/SIAM, 2003, pp. 841–850. [Online]. Available: http://dl.acm.org/citation.cfm?id=644108.644250

[64] P. Ferragina and G. Manzini, "Indexing compressed text," *Journal of the ACM*, vol. 52, no. 4, pp. 552–581, 2005. [Online]. Available: https://doi.org/10.1145/1082036.1082039

[65] R. Grossi and J. S. Vitter, "Compressed suffix arrays and suffix trees with applications to text indexing and string matching," *SIAM Journal on Computing*, vol. 35, no. 2, pp. 378–407, 2005. [Online]. Available: https://doi.org/10.1137/S0097539702402354

[66] K. Sadakane, "Succinct representations of LCP information and improvements in the compressed suffix arrays," in *13th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2002*. ACM/SIAM, 2002, pp. 225–232. [Online]. Available: http://dl.acm.org/citation.cfm?id=545381.545410

[67] S. Gog, T. Beller, A. Moffat, and M. Petri, "From theory to practice: Plug and play with succinct data structures," in *13th International Symposium on Experimental Algorithms, SEA 2014*, ser. LNCS, J. Gudmundsson and J. Katajainen, Eds., vol. 8504. Springer, 2014, pp. 326–337. [Online]. Available: https://doi.org/10.1007/978-3-319-07959-2_28

[68] W. Hon, K. Sadakane, and W. Sung, "Breaking a time-and-space barrier in constructing full-text indices," *SIAM Journal on Computing*, vol. 38, no. 6, pp. 2162–2178, 2009. [Online]. Available: https://doi.org/10.1137/070685373

[69] D. Belazzougui, "Linear time construction of compressed text indices in compact space," in *46th Annual ACM Symposium on Theory of Computing, STOC 2014*, D. B. Shmoys, Ed. ACM, 2014, pp. 148–193. [Online]. Available: https://doi.org/10.1145/2591796.2591885

[70] 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

[71] 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

[72] G. Navarro and V. Mäkinen, "Compressed full-text indexes," *ACM Computing Surveys*, vol. 39, no. 1, pp. 2:1–2:61, 2007. [Online]. Available: https://doi.org/10.1145/1216370.1216372

[73] G. Navarro, "Wavelet trees for all," *Journal of Discrete Algorithms*, vol. 25, pp. 2–20, 2014. [Online]. Available: https://doi.org/10.1016/j.jda.2013.07.004

[74] D. Belazzougui and G. Navarro, "Alphabet-independent compressed text indexing," *ACM Transactions on Algorithms*, vol. 10, no. 4, pp. 23:1–23:19, 2014. [Online]. Available: https://doi.org/10.1145/2635816

[75] G. Navarro, *Compact data structures: A practical approach*. Cambridge, UK: Cambridge University Press, 2016. [Online]. Available: https://doi.org/10.1017/cbo9781316588284

[76] 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

[77] D. Kempa and T. Kociumaka, "Resolution of the Burrows-Wheeler Transform conjecture," 2020, full version of [3]. [Online]. Available: https://arxiv.org/abs/1910.10631v3

[78] 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