# Near-Optimal Search Time in $\delta$-Optimal Space, and Vice Versa*

Tomasz Kociumaka[1], Gonzalo Navarro[2] and Francisco Olivares[2]*

[1]Max Planck Institute for Informatics, Saarbrücken, Germany.
[2]CeBiB — Centre for Biotechnology and Bioengineering
Department of Computer Science, University of Chile, Chile.


*Corresponding author(s). E-mail(s): folivares@uchile.cl;

## Abstract

Two recent lower bounds on the compressibility of repetitive sequences, $\delta \leq \gamma$, have received much attention. It has been shown that a length-$n$ string $S$ over an alphabet of size $\sigma$ can be represented within the optimal $O(\delta \log \frac{n \log \sigma}{\delta \log n})$ space, and further, that within that space one can find all the $occ$ occurrences in $S$ of any length-$m$ pattern in time $O(m \log n + occ \log^\epsilon n)$ for any constant $\epsilon > 0$. Instead, the near-optimal search time $O(m + (occ + 1) \log^\epsilon n)$ has been achieved only within $O(\gamma \log \frac{n}{\gamma})$ space. Both results are based on considerably different locally consistent parsing techniques. The question of whether the better search time could be supported within the $\delta$-optimal space remained open. In this paper, we prove that both techniques can indeed be combined to obtain the best of both worlds: $O(m + (occ + 1) \log^\epsilon n)$ search time within $O(\delta \log \frac{n \log \sigma}{\delta \log n})$ space. Moreover, the number of occurrences can be computed in $O(m + \log^{2+\epsilon} n)$ time within $O(\delta \log \frac{n \log \sigma}{\delta \log n})$ space. We also show that an extra sublogarithmic factor on top of this space enables optimal $O(m + occ)$ search time, whereas an extra logarithmic factor enables optimal $O(m)$ counting time.

**Keywords:** text indexing, pattern matching, substring complexity, repetitive sequences

# 1 Introduction

The amount of data we are expected to handle has been growing steadily in the last decades [1]. The fact that much of the fastest-growing data is composed of highly repetitive sequences has raised interest in text indexes whose size can be bounded by some measure of repetitiveness [2] and in the study of those repetitiveness measures [3]. Since statistical compression does not capture repetitiveness well [4], various other measures have been proposed for this case. Two recent ones, which have received much attention because of their desirable properties, are the size $\gamma$ of the smallest string attractor [5] and a function $\delta$ of the substring complexity [6, 7]. Every string satisfies $\delta \leq \gamma$ [6] (with $\delta = o(\gamma)$ in some string families [7]), and $\gamma$ asymptotically lower-bounds many other measures sensitive to repetitiveness [5] (e.g., the size of the smallest Lempel–Ziv parse [8]). On the other hand, any string $S \in [0..\sigma)^n$ can be represented within $O(\delta \log \frac{n \log \sigma}{\delta \log n})$ space, and this bound is tight for the full spectrum of parameters $n$, $\sigma$, and $\delta$ [7].

A more ambitious goal than merely representing $S$ in compressed space is to *index* it within that space so that, given any pattern $P$, one can efficiently find all the *occ* occurrences of $P$ in $S$. Interestingly, it has been shown that, for any constant $\epsilon > 0$, one can index $S$ within the optimal $O(\delta \log \frac{n \log \sigma}{\delta \log n})$ space and then report the occurrences of any length-$m$ pattern in $O(m \log n + occ \log^\epsilon n)$ time [7]. If one allows the higher $O(\gamma \log \frac{n}{\gamma})$ space, the search time can be reduced to $O(m + (occ+1) \log^\epsilon n)$ [6], which is optimal in terms of the pattern length and near-optimal in the time per reported occurrence. Slightly more space, $O(\gamma \log \frac{n}{\gamma} \log^\epsilon n)$, allows for a truly optimal search time, $O(m + occ)$.[1]

The challenge of obtaining the near-optimal $O(m + (occ + 1) \log^\epsilon n)$ search time within optimal $O(\delta \log \frac{n \log \sigma}{\delta \log n})$ space was posed [6, 7], and this is what we settle on the affirmative in this paper. Both previous results build convenient context-free grammars on $S$ and then adapt a classical grammar-based index on it [9, 10]. The index based on attractors [6] constructs a grammar from a locally consistent parsing [11] of $S$ that forms blocks in $S$ ending at every local minimum with respect to a randomized ordering of the alphabet, collapsing every block into a nonterminal and iterating. The smaller grammar based on substring complexity [7] uses another locally consistent parsing obtained by *recompression* [12], which randomly divides the alphabet into "left" and "right" symbols and combines every left-right pair into a nonterminal, also iterating. The key to obtaining $\delta$-bounded space is to pause the pairing on symbols whose expansions become too long for the iteration where they were formed [7]. We show that the pausing idea can be applied to the first kind of locally consistent grammar as well so that it yields the desired time and space complexities. The next theorem summarizes our first result.

---

[1]In this work, we assume that $P[1..m]$ is represented in $O(m)$ space. For small alphabets, the packed setting, where $P$ occupies $O(\lceil \frac{m \log \sigma}{\log n} \rceil)$ space, could also be considered; see [3, Sec. 2.2].

**Theorem 1.1.** *For every constant $\epsilon > 0$, given a string $S \in [0 .. \sigma)^n$ with measure $\delta$, one can build in $O(n)$ expected time a data structure using $O(\delta \log \frac{n \log \sigma}{\delta \log n})$ words of space such that, later, given a pattern $P[1 .. m]$, one can find all of its occ occurrences in $S$ in $O(m + \log^\epsilon \delta + occ \log^\epsilon(\delta \log \frac{n \log \sigma}{\delta \log n})) \subseteq O(m + (occ + 1) \log^\epsilon n)$ time.*

Apart from this near-optimal search time within optimal space, we can obtain optimal search time within near-optimal space, increasing the space by only a sublogarithmic factor.

**Theorem 1.2.** *For every constant $\epsilon > 0$, given a string $S \in [0 .. \sigma)^n$ with measure $\delta$, one can build in $O(n + \delta \log \frac{n \log \sigma}{\delta \log n} \log n)$ expected time a data structure using $O(\delta \log \frac{n \log \sigma}{\delta \log n} \log^\epsilon(\delta \log \frac{n \log \sigma}{\delta \log n})) \subseteq O(\delta \log \frac{n \log \sigma}{\delta \log n} \log^\epsilon n)$ words of space such that, later, given a pattern $P[1 .. m]$, one can find all of its occ occurrences in $S$ in optimal time $O(m + occ)$.*

Finally, we show how to efficiently count the number of occurrences of a pattern in $S$ within $\delta$-bounded space, while matching the times of the previous $\gamma$-bounded space structure.

**Theorem 1.3.** *For every constant $\epsilon > 0$, given a string $S \in [0 .. \sigma)^n$ with measure $\delta$, one can build in $O(n \log n)$ expected time a data structure using $O(\delta \log \frac{n \log \sigma}{\delta \log n})$ words of space such that, later, given a pattern $P[1 .. m]$, one can compute the number occ of its occurrences in $S$ in time $O(m + \log^{2+\epsilon} n)$. We can also build, in $O(n \log n)$ expected time, a data structure using $O(\delta \log \frac{n \log \sigma}{\delta \log n} \log(\delta \log \frac{n \log \sigma}{\delta \log n})) \subseteq O(\delta \log \frac{n \log \sigma}{\delta \log n} \log n)$ words of space that computes occ in optimal time $O(m)$.*

Our algorithms are designed for the (standard) RAM model of computation with machine words of $w = \Theta(\log n)$ bits each. By default, we measure the space in words, which means that $O(x)$ space comprises $O(x \log n)$ bits.

A conference version of this paper appeared in *Proc. LATIN 2022* [13]. The current version provides more refined space results, which incorporate the alphabet size $\sigma$ and reach optimality for every $n$, $\delta$, and $\sigma$ [7]. It also includes the results on counting the number of occurrences and achieves optimal search times within near-optimal space.

## 2 Notation and Basic Concepts

A string is a sequence $S[1 .. n] = S[1] \cdot S[2] \cdots S[n]$ of symbols, where each symbol belongs to an alphabet $\Sigma = [0 .. \sigma) = \{0, \ldots, \sigma - 1\}$. We denote as $\Sigma(S)$ the subset of $\Sigma$ consisting of symbols that occur in $S$. The length of $S$ is denoted $|S| = n$. The only string of length zero is denoted $\varepsilon$, so $|\varepsilon| = 0$. We assume that the alphabet size is a polynomial function of $n$, that is, $\sigma = n^{O(1)}$. The concatenation of strings $S$ and $S'$ is denoted $S \cdot S' = SS'$. A string $S'$ is

a substring of $S$ if $S' = S(i \mathinner{.\,.} j] = S[i+1] \cdots S[j]$ for some $0 \le i \le j \le n$; in particular $S(i \mathinner{.\,.} j] = \varepsilon$ if $j = i$. With the term *fragment*, we refer to a particular occurrence $S(i \mathinner{.\,.} j]$ of a substring in $S$ (not just the substring content). We also use other combinations of round and square brackets to denote fragments: $S(i \mathinner{.\,.} j) = S(i \mathinner{.\,.} j-1]$, $S[i \mathinner{.\,.} j] = S(i-1 \mathinner{.\,.} j]$, and $S[i \mathinner{.\,.} j) = S(i-1 \mathinner{.\,.} j-1]$. We use $S^{rev}$ to denote the reverse of $S$, that is, $S^{rev} = S[n] \cdot S[n-1] \cdots S[1]$.

A *straight line program* (SLP) is a context-free grammar where each nonterminal appears once at the left-hand side of a rule, and where the non-terminals can be sorted so that the right-hand sides refer to terminals and preceding nonterminals. Such an SLP generates a single string. Furthermore, we refer to a *run-length straight line program* (RLSLP) as an SLP that, in addition, allows rules of the form $A \to A_1^s$, where $A, A_1$ are nonterminals and $s \in \mathbb{Z}_{\ge 2}$, which means that the right-hand side of the rule defining $A$ can be obtained by concatenating $s$ copies of $A_1$.

A *parsing* is a way to decompose a string $S$ into non-overlapping *blocks*, $S = S_1 \cdot S_2 \cdots S_k$. A *locally consistent parsing (LCP)* [14] is a parsing where, if two positions $i, i'$ have long enough matching contexts $S[i - \alpha \mathinner{.\,.} i + \beta] = S[i' - \alpha \mathinner{.\,.} i' + \beta]$ and there is a block boundary after $S[i]$, then there is also one after $S[i']$. The meaning of "long enough" depends on the LCP type [6, 14, 15].

# 3 A New *δ*-bounded RLSLP

The measure $\delta$ was implicitly used in a stringology context already in [16], but it was formally defined later [6] (as a way to construct an RLSLP of size $O(\gamma \log \frac{n}{\gamma})$ without knowing $\gamma$) and thoroughly studied in [7]. For a given string $S$ and integer $k \ge 0$, let $d_k(S)$ be the number of distinct length-$k$ substrings in $S$. The sequence of all values $d_k(S)$ is known as the *substring complexity* of $S$. Then, $\delta$ is defined as

$$\delta \;=\; \max\left\{ \frac{d_k(S)}{k} : k \in \mathbb{Z}_{\ge 1} \right\}.$$

The bounds of [7], including $\gamma = O(\delta \log \frac{n \log \sigma}{\delta \log n})$, implicitly utilize the following:

**Fact 3.1.** *Every string $S \in [0 \mathinner{.\,.} \sigma)^n$ of measure $\delta$ satisfies*

$$\sum_{p=0}^{\infty} \frac{d_{2^p}(S)}{2^p} \le 5\delta + \delta \log \frac{n \log \sigma}{\delta \log n} = O(\delta \log \frac{n \log \sigma}{\delta \log n}).$$

*Proof* Let $\mu := \lfloor \log \frac{\log \delta}{\log \sigma} \rfloor$ and $\nu := \lceil \log \frac{n}{\delta} \rceil$. For $p \in [0 \mathinner{.\,.} \mu]$, we use a bound $d_{2^p}(S) \le \sigma^{2^p} \le \sigma^{2^\mu}$ to derive

$$\sum_{p=0}^{\mu} \frac{d_{2^p}(S)}{2^p} \le \sum_{p=0}^{\mu} \frac{\sigma^{2^\mu}}{2^p} \le 2 \cdot \sigma^{2^\mu} \le 2 \cdot \sigma^{\frac{\log \delta}{\log \sigma}} = 2\delta.$$

For $p \in (\mu \mathbin{.\,.} \nu)$, we use a bound $d_{2^p}(S) \leq \delta \cdot 2^p$ to derive

$$\sum_{p=\mu+1}^{\nu-1} \frac{d_{2^p}(S)}{2^p} \leq \delta \cdot (\nu - \mu - 1) \leq \delta \cdot \left(1 + \log \tfrac{n}{\delta} - \log \tfrac{\log \delta}{\log \sigma}\right) = \delta + \delta \log \tfrac{n \log \sigma}{\delta \log \delta}.$$

For $p \geq \nu$, we use a bound $d_{2^p}(S) \leq n$ to derive

$$\sum_{p=\nu}^{\infty} \frac{d_{2^p}(S)}{2^p} \leq \sum_{p=\nu}^{\infty} \frac{n}{2^p} \leq \frac{2n}{2^\nu} \leq 2\delta.$$

Overall, we get $\sum_{p=0}^{\infty} \frac{d_{2^p}(S)}{2^p} \leq 2\delta + \delta + \delta \log \frac{n \log \sigma}{\delta \log \delta} + 2\delta = O(\delta \log \frac{n \log \sigma}{\delta \log n})$, where we note that $\delta \log \frac{n \log \sigma}{\delta \log \delta} = O(\delta \log \frac{n \log \sigma}{\delta \log n})$.[2] □

An RLSLP of size $O(\delta \log \frac{n \log \sigma}{\delta \log n})$ was built [7] on top of the recompression method [12]. In this section, we show that the same can be achieved on top of the block-based LCP [11]. Unlike the previous construction, ours produces an RLSLP with $O(\delta \log \frac{n \log \sigma}{\delta \log n})$ rules in $O(n)$ deterministic time, though we still use randomization to ensure that the total grammar size is also $O(\delta \log \frac{n \log \sigma}{\delta \log n})$.

We adapt the preceding construction [7], which uses the so-called *restricted recompression* [17]. This technique pauses the processing for symbols whose expansion is too long for the current stage. A similar idea was used [18, 19] for adapting another LCP, called *signature parsing* [20]. We apply restriction (the pausing technique) to the LCP of [11] that forms blocks ending at local minima with respect to a randomized ordering of the alphabet. This LCP will be used later to obtain near-optimal search time, extending previous work [6]. We call our parsing *restricted block compression*.

## 3.1 Restricted block compression

Given a string $S \in \Sigma^+$, our restricted block compression builds a sequence of strings $(S_k)_{k \geq 0}$ over the alphabet $\mathcal{A}$ defined recursively to contain symbols in $\Sigma$, pairs formed by a symbol in $\mathcal{A}$ and an integer $m \geq 2$, and sequences of at least two symbols in $\mathcal{A}$; formally, $\mathcal{A}$ is the least fixed point of the expression:

$$\mathcal{A} = \Sigma \cup (\mathcal{A} \times \mathbb{Z}_{\geq 2}) \cup \bigcup_{i=2}^{\infty} \mathcal{A}^i.$$

In the following, we denote $\bigcup_{i=2}^{\infty} \mathcal{A}^i$ with $\mathcal{A}^{\geq 2}$.

Symbols in $\mathcal{A} \setminus \Sigma$ are *nonterminals*, which are naturally associated with *productions* $(A_1, \ldots, A_s) \to A_1 \cdots A_s$ for $(A_1, \ldots, A_s) \in \mathcal{A}^{\geq 2}$ and $(A_1, s) \to A_1^s$ for $(A_1, s) \in \mathcal{A} \times \mathbb{Z}_{\geq 2}$. Setting any $A \in \mathcal{A}$ as the starting symbol yields an RLSLP. The string generated by this RLSLP is $\mathsf{exp}(A)$, where $\mathsf{exp} : \mathcal{A} \to \Sigma^+$

---

[2] If $\delta \log n \leq \sqrt{n}$, then $\delta \log \frac{n \log \sigma}{\delta \log \delta} \leq \delta \log(n \log \sigma) \leq 2\delta \log(\sqrt{n} \log \sigma) \leq 2\delta \log \frac{n \log \sigma}{\delta \log n}$. Otherwise, $\log \delta > \frac{1}{2} \log n - \log \log n > \frac{1}{4} \log n$, so $\delta \log \frac{n \log \sigma}{\delta \log \delta} \leq \delta \log \frac{4n \log \sigma}{\delta \log n} = \delta \log \frac{n \log \sigma}{\delta \log n} + 2\delta$.

is the *expansion* function defined recursively:

$$\exp(A) = \begin{cases} A & \text{if } A \in \Sigma, \\ \exp(A_1) \cdots \exp(A_s) & \text{if } A = (A_1, \ldots, A_s) \text{ for } A_1, \ldots, A_s \in \mathcal{A}, \\ \exp(A_1)^s & \text{if } A = (A_1, s) \text{ for } A_1 \in \mathcal{A} \text{ and } s \in \mathbb{Z}_{\geq 2}. \end{cases}$$

The expansion function is extended homomorphically to $\exp : \mathcal{A}^* \to \Sigma^*$, with $\exp(A_1 \cdots A_m) = \exp(A_1) \cdots \exp(A_m)$ for $A_1 \cdots A_m \in \mathcal{A}^*$.

Starting from $S_0 = S$, the strings $(S_k)_{k=0}^{\infty}$ such that $\exp(S_k) = S$ are built by the alternate applications of two functions, both of which decompose a string $T \in \mathcal{A}^+$ into *blocks* (by placing *block boundaries* between some characters) and then collapse blocks of length $s \geq 2$ into individual symbols in $\mathcal{A}$. In Definition 3.2, the blocks are maximal *runs* of the same symbol in a subset $\mathcal{B} \subseteq \mathcal{A}$, and they are collapsed to symbols in $\mathcal{A} \times \mathbb{Z}_{\geq 2}$.

**Definition 3.2** (Run-length encoding). Given $T \in \mathcal{A}^+$ and a subset of symbols $\mathcal{B} \subseteq \mathcal{A}$, we define $rle_{\mathcal{B}}(T) \in \mathcal{A}^+$ as the string obtained by decomposing $T$ into blocks and collapsing these blocks as follows:
1) For every $i \in [1 \mathinner{.\,.} |T|)$, place a block boundary between $T[i]$ and $T[i+1]$ if $T[i] \notin \mathcal{B}$, $T[i+1] \notin \mathcal{B}$, or $T[i] \neq T[i+1]$.
2) For every block $T[i \mathinner{.\,.} i+s)$ of $s \geq 2$ equal symbols $A$, replace $T[i \mathinner{.\,.} i+s) = A^s$ with the symbol $(A, s) \in \mathcal{A}$.

In Definition 3.4, the block boundaries are determined by the local minima of a function of the set $\Sigma(T) \subseteq \mathcal{A}$ of symbols that occur in $T$, and the blocks are collapsed to symbols in $\mathcal{A}^{\geq 2}$.

**Definition 3.3** (Local minima). Given $T \in \mathcal{A}^+$, we say that a position $i \in (1 \mathinner{.\,.} |T|)$ is a *local minimum* with respect to a function $\pi : \Sigma(T) \to \mathbb{Z}$ if

$$\pi(T[i-1]) > \pi(T[i]) \text{ and } \pi(T[i]) < \pi(T[i+1]).$$

**Definition 3.4** (Restricted block parsing). Given $T \in \mathcal{A}^+$, a function $\pi : \Sigma(T) \to \mathbb{Z}$, and a subset of symbols $\mathcal{B} \subseteq \mathcal{A}$, we define $bc_{\pi, \mathcal{B}}(T) \in \mathcal{A}^+$ as the string obtained by decomposing $T$ into blocks and collapsing these blocks as follows:
1) For every $i \in [1 \mathinner{.\,.} |T|)$, place a block boundary between $T[i]$ and $T[i+1]$ if $T[i] \notin \mathcal{B}$, $T[i+1] \notin \mathcal{B}$, or $i$ is a local minimum with respect to $\pi$.
2) For each block $T[i \mathinner{.\,.} i+s)$ of length $s \geq 2$, replace $T[i \mathinner{.\,.} i+s)$ with a symbol $(T[i], \ldots, T[i+s-1]) \in \mathcal{A}$.

Note that $\mathcal{B}$ consists of *active* symbols that can be combined into larger blocks; we say that the other symbols are *paused*. The idea of our restricted block compression is to create successive strings $S_k$, starting from $S_0 = S$. At the odd levels $k$, we perform run-length encoding on the preceding string $S_{k-1}$.

On the even levels $k$, we perform block parsing on the preceding string $S_{k-1}$. We pause the symbols whose expansions are too long for that level.

**Definition 3.5** (Restricted block compression). Given $S \in \Sigma^+$, the strings $S_k$ for $k \in \mathbb{Z}_{\geq 0}$ are constructed as follows, where $\ell_k := \left(\frac{4}{3}\right)^{\lceil k/2 \rceil - 1}$, $\mathcal{A}_k := \{A \in \mathcal{A} : |\exp(A)| \leq \ell_k\}$, and $\pi_k : \Sigma(S_{k-1}) \to [1 \mathinner{.\,.} |\Sigma(S_{k-1})|]$ is a bijection satisfying $\pi_k(A) < \pi_k(B)$ for $A \in \Sigma(S_{k-1}) \setminus \mathcal{A}_k$ and $B \in \Sigma(S_{k-1}) \cap \mathcal{A}_k$:
- If $k = 0$, then $S_k = S$.
- If $k > 0$ is odd, then $S_k = rle_{\mathcal{A}_k}(S_{k-1})$.
- If $k > 0$ is even, then $S_k = bc_{\pi_k, \mathcal{A}_k}(S_{k-1})$.

## 3.2 Grammar size analysis

Our RLSLP will be built by performing restricted block compression as long as $|S_k| > 1$. Although the resulting RLSLP formally has infinitely many symbols, we can remove those having no occurrences in any $S_k$. To define the actual symbols in the grammar, for all $k \in \mathbb{Z}_{\geq 0}$, denote $\mathcal{S}_k := \{S_k[j] : j \in [1 \mathinner{.\,.} |S_k|]\}$ and $\mathcal{S} := \bigcup_{k=0}^{\infty} \mathcal{S}_k$.

Recall that $\exp(S_k) = S$ holds for every $k \in \mathbb{Z}_{\geq 0}$. Based on this, we associate $S_k$ with a decomposition of $S$ into *phrases*.

**Definition 3.6** (Phrase boundaries). For every $k \in \mathbb{Z}_{\geq 0}$ and $j \in [1 \mathinner{.\,.} |S_k|]$, we define the level-$k$ *phrases* of $S$ induced by $S_k$ as the fragments

$$S(|\exp(S_k[1 \mathinner{.\,.} j))| \mathinner{.\,.} |\exp(S_k[1 \mathinner{.\,.} j])|] = \exp(S_k[j]).$$

We also define the set $B_k \subseteq [0 \mathinner{.\,.} n]$ of *phrase boundaries* induced by $S_k$:

$$B_k = \{|\exp(S_k[1 \mathinner{.\,.} j])| : j \in [0 \mathinner{.\,.} |S_k|]\}.$$

Note that taking $j = 0$ and $j = |S_k|$ yields $0 \in B_k$ and $n \in B_k$, respectively.

The following lemma captures the local consistency of restricted block compression: the phrase boundaries are determined by a small context.

**Lemma 3.7.** *Consider integers $k, m, \alpha \in \mathbb{Z}_{\geq 0}$ with $\alpha \geq \lfloor 8\ell_k \rfloor$ and $i, i' \in [m + 2\alpha \mathinner{.\,.} n - \alpha]$ such that $S(i - m - 2\alpha \mathinner{.\,.} i + \alpha] = S(i' - m - 2\alpha \mathinner{.\,.} i' + \alpha]$.*
1) *If $i \in B_k$, then $i' \in B_k$.*
2) *If $S(i - m \mathinner{.\,.} i]$ is a level-$k$ phrase, then $S(i' - m \mathinner{.\,.} i']$ is a level-$k$ phrase corresponding to the same symbol in $S_k$.*

*Proof* We proceed by induction on $k$, with a weaker assumption $\alpha \geq \lfloor 7\ell_k \rfloor$ for odd $k$. In the base case of $k = 0$, the claim is trivial because $B_0 = [0 \mathinner{.\,.} n]$ and every character of $S$ forms a level-0 phrase. Next, we prove that the claim holds for integers $k > 0$ and $\alpha > \lfloor \ell_k \rfloor$ assuming that it holds for $k - 1$ and $\alpha - \lfloor \ell_k \rfloor$. This is sufficient for

the inductive step: If $\alpha \geq \lfloor 8\ell_k \rfloor$ for even $k > 0$, then $\alpha - \lfloor \ell_k \rfloor \geq \lfloor 7\ell_k \rfloor = \lfloor 7\ell_{k-1} \rfloor$. Similarly, if $\alpha \geq \lfloor 7\ell_k \rfloor$ for odd $k$, then $\alpha - \lfloor \ell_k \rfloor \geq \lfloor 6\ell_k \rfloor = \lfloor 8\ell_{k-1} \rfloor$.

We start with the first item, where we can assume $m = 0$ without loss of generality. For a proof by contradiction, suppose that $S(i - 2\alpha .. i + \alpha] = S(i' - 2\alpha .. i' + \alpha]$ and $i \in B_k$ yet $i' \notin B_k$. By the first item of the inductive assumption applied to positions $i$ and $i'$, we conclude that $i \in B_k \subseteq B_{k-1}$ implies $i' \in B_{k-1}$. Let us set $j, j' \in [1 .. |S_{k-1}|)$ so that $i = |\exp(S_{k-1}[1 .. j])|$ and $i' = |\exp(S_{k-1}[1 .. j'])|$. Since $i \in B_k$ yet $i' \notin B_k$, the parsing of $S_{k-1}$ places a block boundary between $S_{k-1}[j]$ and $S_{k-1}[j + 1]$, but it does not place a block boundary between $S_{k-1}[j']$ and $S_{k-1}[j' + 1]$. By Definitions 3.2 and 3.4, the latter yields $S_{k-1}[j'], S_{k-1}[j' + 1] \in \mathcal{A}_k$. Consequently, the level-$(k - 1)$ phrases $S(i' - \ell .. i'] := \exp(S_{k-1}[j'])$ and $S(i' .. i' + r] := \exp(S_{k-1}[j' + 1])$ around position $i'$ are of length at most $\lfloor \ell_k \rfloor$ each. Therefore, the assumption $S(i - 2\alpha .. i + \alpha] = S(i' - 2\alpha .. i' + \alpha]$ implies $S(i' - \ell - 2(\alpha - \lfloor \ell_k \rfloor) .. i' + (\alpha - \lfloor \ell_k \rfloor)] = S(i - \ell - 2(\alpha - \lfloor \ell_k \rfloor) .. i + (\alpha - \lfloor \ell_k \rfloor)]$ as well as $S((i' + r) - r - 2(\alpha - \lfloor \ell_k \rfloor) .. i' + r + (\alpha - \lfloor \ell_k \rfloor)] = S((i + r) - r - 2(\alpha - \lfloor \ell_k \rfloor) .. i + r + (\alpha - \lfloor \ell_k \rfloor)]$. Thus, we can apply (the second item of) the inductive assumption for the level-$(k - 1)$ phrases $S(i' - \ell .. i']$ and $S(i' .. i' + r]$, concluding that $S(i - \ell .. i]$ and $S(i .. i + r]$ are also level-$(k - 1)$ phrases parsed into $S_{k-1}[j] = S_{k-1}[j']$ and $S_{k-1}[j + 1] = S_{k-1}[j' + 1]$, respectively.

If $k$ is odd, then a boundary between two symbols in $\mathcal{A}_k$ is placed if and only if the two symbols differ. Consequently, $S_{k-1}[j'] = S_{k-1}[j' + 1]$ and $S_{k-1}[j] \neq S_{k-1}[j + 1]$. This contradicts $S_{k-1}[j] = S_{k-1}[j']$ and $S_{k-1}[j + 1] = S_{k-1}[j' + 1]$.

Thus, it remains to consider the case of even $k$. Since the block parsing placed a boundary between $S_{k-1}[j], S_{k-1}[j + 1] \in \mathcal{A}_k$, we conclude from Definition 3.4 that $j$ must be a local minimum with respect to $\pi_k$, i.e., $\pi_k(S_{k-1}[j - 1]) > \pi_k(S_{k-1}[j]) < \pi_k(S_{k-1}[j + 1])$. Due to $S_{k-1}[j] \in \mathcal{A}_k$, the condition on $\pi_k$ imposed in Definition 3.5 implies $S_{k-1}[j - 1] \in \mathcal{A}_k$. Consequently, the level-$(k - 1)$ phrase $S(i - \ell - \ell' .. i - \ell] := \exp(S_{k-1}[j - 1])$ is of length at most $\lfloor \ell_k \rfloor$, and thus $\ell + \ell' \leq 2\lfloor \ell_k \rfloor$. Therefore, the assumption $S(i - 2\alpha .. i + \alpha] = S(i' - 2\alpha .. i' + \alpha]$ implies $S((i - \ell) - \ell' - 2(\alpha - \lfloor \ell_k \rfloor) .. (i - \ell) + (\alpha - \lfloor \ell_k \rfloor)] = S((i' - \ell) - \ell' - 2(\alpha - \lfloor \ell_k \rfloor) .. (i' - \ell) + (\alpha - \lfloor \ell_k \rfloor)]$. Hence, we can apply (the second item of) the inductive assumption for the level-$(k - 1)$ phrase $S(i - \ell - \ell' .. i - \ell]$, concluding that $S(i' - \ell - \ell' .. i' - \ell]$ is a level-$(k - 1)$ phrase parsed into $S_{k-1}[j' - 1] = S_{k-1}[j - 1]$. Thus, $\pi_k(S_{k-1}[j' - 1]) = \pi_k(S_{k-1}[j - 1]) > \pi_k(S_{k-1}[j']) = \pi_k(S_{k-1}[j]) < \pi_k(S_{k-1}[j' + 1]) = \pi_k(S_{k-1}[j + 1])$, which means that $j'$ is a local minimum with respect to $\pi_k$ and contradicts $i' \notin B_k$ by Definition 3.4.

Let us proceed to the proof of the second item. Let $S_{k-1}(j - m' .. j]$ be the block corresponding to the level-$k$ phrase $S(i - m .. i]$. By the inductive assumption applied separately to every level-$(k - 1)$ phrase contained in $S(i - m .. i]$, the fragment $S(i' - m .. i']$ consists of level-$(k - 1)$ phrases that, in $S_{k-1}$, are collapsed into a fragment $S_{k-1}(j' - m' .. j']$ matching $S_{k-1}(j - m' .. j]$. Moreover, by the already proved first item, applied to corresponding positions within $[i - m .. i]$ and $[i' - m .. i']$, the parsing of $S_{k-1}$ places block boundaries around $S_{k-1}(j' - m'..j']$ but nowhere within that fragment. Hence, $S_{k-1}(j - m' .. j]$ and $S_{k-1}(j' - m' .. j']$ are matching blocks, which means that they are collapsed into matching symbols of $S_k$. The phrases $S(i - m .. i]$ and $S(i' - m .. i']$ are thus represented by matching symbols in $S_k$.  □

Our next goal is to prove that the phrase boundaries are locally sparse, that is, their number within any interval $I$ decreases exponentially with $k$. This holds because two out of any three consecutive blocks must be merged in the even levels, unless one is already long for that level (and thus paused).

**Lemma 3.8.** *For every $k \in \mathbb{Z}_{\geq 0}$ and non-empty interval $I \subseteq [0 \mathrel{..} n]$, we have*

$$|B_k \cap I| < 2 + \tfrac{4|I|-4}{\ell_{k+1}}.$$

*Proof* We proceed by induction on $k$. For $k = 0$, we have $|B_k \cap I| = |I| < 2+4|I|-4 = 2 + \frac{4|I|-4}{\ell_1}$. If $k$ is odd, we note that $\ell_{k+1} = \ell_k$ and $B_k \subseteq B_{k-1}$. Consequently,

$$|B_k \cap I| \leq |B_{k-1} \cap I| < 2 + \tfrac{4|I|-4}{\ell_k} = 2 + \tfrac{4|I|-4}{\ell_{k+1}}.$$

If $k$ is even, let us define

$$J = \{j \in [1 \mathrel{..} |S_{k-1}|) : |\exp(S_{k-1}[1 \mathrel{..} j-1])| \in I \text{ and } |\exp(S_{k-1}[1 \mathrel{..} j+1])| \in I\},$$

$$J' = \{j \in J : S_{k-1}[j] \in \mathcal{A}_k \text{ and } S_{k-1}[j+1] \in \mathcal{A}_k\},$$

Observe that $|J| = \max(0, |B_{k-1} \cap I| - 2)$: every $j \in J$ corresponds to a position $|\exp(S_{k-1}[1 \mathrel{..} j])| \in B_{k-1} \cap I$ which is neither the leftmost nor the rightmost one in $B_{k-1} \cap I$. If $j \notin J'$, then $S_{k-1}[j] \notin \mathcal{A}_k$ or $S_{k-1}[j+1] \notin \mathcal{A}_k$. The level-$(k-1)$ phrase $S(p \mathrel{..} q]$ corresponding to this pausing symbol satisfies $[p \mathrel{..} q] \subseteq I$ and, by definition of $\mathcal{A}_k$, it is of length $q - p > \ell_k$. There are at most $\frac{|I|-1}{\ell_k}$ phrases $S(p \mathrel{..} q]$ satisfying both conditions, and thus $|J \setminus J'| \leq 2 \cdot \frac{|I|-1}{\ell_k}$.

Every position $i \in B_k \cap I$, except for the leftmost and the rightmost one, corresponds to a position $j \in J$ such that $i = |\exp(S_{k-1}[1 \mathrel{..} j])|$ and the restricted block parsing places a block boundary between $S_{k-1}[j]$ and $S_{k-1}[j+1]$. If $j \in J'$, then $j$ must be a local minimum with respect to $\pi_k$ and, in particular, $\pi_k(S_{k-1}[j-1]) > \pi_k(S_{k-1}[j])$. By the condition on $\pi_k$ specified in Definition 3.5, this implies that $S_{k-1}[j-1] \in \mathcal{A}_k$. Thus, the restricted block parsing does not place a block boundary between $S_{k-1}[j-1]$ and $S_{k-1}[j]$. Since $i$ is not the leftmost position in $B_k \cap I$, we must have $j - 1 \in J'$. Overall, using the inductive assumption, we conclude that

$$|B_k \cap I| \leq 2 + |J \setminus J'| + \tfrac{1}{2}|J'| = 2 + \tfrac{1}{2}|J \setminus J'| + \tfrac{1}{2}|J|$$

$$= 2 + \tfrac{1}{2}|J \setminus J'| + \max\left(0, \tfrac{1}{2}|B_{k-1} \cap I| - 1\right) < 2 + \tfrac{|I|-1}{\ell_k} + \tfrac{2|I|-2}{\ell_k} = 2 + \tfrac{4|I|-4}{\ell_{k+1}}.$$

$\square$

Plugging $I = [0 \mathrel{..} n]$, we derive an upper bound on $|S_k| = |B_k| - 1$.

**Corollary 3.9.** *For every $k \in \mathbb{Z}_{\geq 0}$, we have $|S_k| < 1 + \frac{4n}{\ell_{k+1}}$. Consequently, $|S_\kappa| = 1$ holds for $\kappa := 2\lceil \log_{4/3}(4n) \rceil$.*

The following lemma essentially bounds the substring complexity of each string $S_k$ in terms of the substring complexity of $S$. An important detail, however, is that we only consider substrings of $S_k$ consisting of active symbols (in $\mathcal{A}_{k+1}$). This is sufficient because the pausing symbols form length-1 blocks that are copied to $S_{k+1}$ without getting collapsed.

**Lemma 3.10.** *For all integers $k \geq 0$ and $m \geq 1$, the number of distinct substrings of $S_k$ belonging to $\mathcal{A}_{k+1}^m$ is $O(m(1 + \frac{d_r(S)}{r}))$, where $r = 2^{\lceil \log(50m\ell_{k+1}) \rceil}$.*

*Proof* Denote $\alpha := \lfloor 8\ell_k \rfloor$ and $\ell := \frac{r}{2} \geq 25m\ell_{k+1} \geq 3\alpha + m\lfloor \ell_{k+1} \rfloor$, and let $L$ be the set consisting of the trailing $\ell$ positions in $S$ and all positions covered by the leftmost occurrences of substrings of $S$ of length at most $\ell$. We first prove two auxiliary claims.

**Claim 3.11.** *The string $S_k$ contains at most $|L \cap B_k|$ distinct substrings in $\mathcal{A}_{k+1}^m$.*

*Proof* Let us fix a substring $T \in \mathcal{A}_{k+1}^m$ of $S_k$ and let $S_k(j - m \mathinner{.\,.} j]$ be the leftmost occurrence of $T$ in $S_k$. Moreover, let $p = |\exp(S_k[1 \mathinner{.\,.} j - m])|$ and $q = |\exp(S_k[1 \mathinner{.\,.} j])|$ so that $S(p \mathinner{.\,.} q]$ is the expansion of $S_k(j - m \mathinner{.\,.} j]$. By $S_k(j - m \mathinner{.\,.} j] \in \mathcal{A}_{k+1}^m$, we have $q - p \leq m\lfloor \ell_{k+1} \rfloor \leq \ell - 3\alpha$.

Suppose that $q \notin L$. Due to $(0 \mathinner{.\,.} \ell] \cup (n - \ell \mathinner{.\,.} n] \subseteq L$, this implies that $q \in (\ell \mathinner{.\,.} n - \ell]$ is not covered by the leftmost occurrence of any substring of length at most $\ell$. In particular, $S(p - 2\alpha \mathinner{.\,.} q + \alpha]$ must have an earlier occurrence $S(p' - 2\alpha \mathinner{.\,.} q' + \alpha]$ for some $p' < p$ and $q' < q$. Consequently, Lemma 3.7, applied to subsequent level-$k$ phrases comprising $S(p \mathinner{.\,.} q]$, shows that $S(p' \mathinner{.\,.} q']$ consists of full level-$k$ phrases and the corresponding fragment of $S_k$ matches $S_k(j - m \mathinner{.\,.} j] = T$. By $q' < q$, this contradicts the assumption that $S_k(j - m \mathinner{.\,.} j]$ is the leftmost occurrence of $T$ in $S_k$.

Thus, we must have $q \in L$. A level-$k$ phrase ends at position $q$, so we also have $q \in B_k$. Since the position $q$ uniquely determines the substring $T \in \mathcal{A}_{k+1}^m$, this yields an upper bound of $|L \cap B_k|$ on the number of choices for $T$. $\qquad\square$

**Claim 3.12.** *The set $L$ forms $O(1 + \frac{d_r(S)}{r})$ intervals of total length $O(r + d_r(S))$.*

*Proof* Each position in $L \cap (0 \mathinner{.\,.} n - \ell]$ is covered by the leftmost occurrence of a length-$\ell$ substring of $S$, and thus $L$ forms at most $\lfloor \frac{1}{\ell}|L| \rfloor$ intervals of length at least $\ell$ each. Hence, it suffices to prove that the total length satisfies $|L| = O(r + d_r(S))$. For this, note that, for each position $j \in L \cap [\ell \mathinner{.\,.} n - \ell]$, the fragment $S(j - \ell \mathinner{.\,.} j + \ell]$ is the leftmost occurrence of a length-$r$ substring of $S$; this because any length-$\ell$ fragment covering position $j$ is contained within $S(j - \ell \mathinner{.\,.} j + \ell]$. Consequently, $|L| \leq r + d_r(S)$ holds as claimed. $\qquad\square$

By Claim 3.11, it remains to bound $|L \cap B_k|$. Let $\mathcal{I}$ be the family of intervals covering $L$. For each $I \in \mathcal{I}$, Lemma 3.8 implies $|B_k \cap I| \leq 1 + \frac{4|I|}{\ell_{k+1}}$. By the bounds on $\mathcal{I}$ following from Claim 3.12, this yields the following result:

$$|B_k \cap L| \leq |\mathcal{I}| + \frac{4}{\ell_{k+1}} \sum_{I \in \mathcal{I}} |I| = O\left(1 + \frac{d_r(S)}{r} + \frac{r + d_r(S)}{\ell_{k+1}}\right) = O\left(m\left(1 + \frac{d_r(S)}{r}\right)\right),$$

where the last inequality holds because $r = \Theta(m\ell_{k+1})$. $\qquad\square$

The following result combines Fact 3.1, Corollary 3.9, and Lemma 3.10.

**Corollary 3.13.** *For every string $S \in [0 \mathinner{.\,.} \sigma)^n$ with measure $\delta$, we have $|\mathcal{S}| = O(\delta \log \frac{n \log \sigma}{\delta \log n})$.*

*Proof* Note that $|\mathcal{S}| \leq 1 + \sum_{k=0}^{\kappa - 1} |\mathcal{S}_k \setminus \mathcal{S}_{k+1}|$. First, we observe that Definition 3.5 guarantees $\mathcal{S}_k \setminus \mathcal{S}_{k+1} \subseteq \mathcal{S}_k \cap \mathcal{A}_{k+1}$. Moreover, each symbol in $\mathcal{S}_k \cap \mathcal{A}_{k+1}$ corresponds to a distinct character of $S_k$ contained in $\mathcal{A}_{k+1}$, and thus $|\mathcal{S}_k \setminus \mathcal{S}_{k+1}| \leq |\mathcal{S}_k \cap \mathcal{A}_{k+1}| =$

$O\big(1 + \frac{d_{r_k}(S)}{r_k}\big)$ holds due to Lemma 3.10 for $r_k := 2^{\lceil \log(50\ell_{k+1}) \rceil}$. Observe that, for every $p \in [0 \mathinner{.\,.} \lceil \log(50\ell_\kappa) \rceil]$, there are at most 6 integers $k$ such that $r_k = 2^p$ (this is because $\ell_{k+6} = (4/3)^3 \ell_k > 2\ell_k$). Hence, Fact 3.1 and the definition of $\kappa$ yield

$$\sum_{k=0}^{\kappa-1} |\mathcal{S}_k \cap \mathcal{A}_{k+1}| = \sum_{p=0}^{\lceil \log(50\ell_\kappa) \rceil} O\Big(1 + \frac{d_{2^p}(S)}{2^p}\Big) = O(\log n + \delta \log \tfrac{n \log \sigma}{\delta \log n}) = O(\delta \log \tfrac{n \log \sigma}{\delta \log n}).$$

Overall, $|\mathcal{S}| = 1 + O(\delta \log \frac{n \log \sigma}{\delta \log n}) = O(\delta \log \frac{n \log \sigma}{\delta \log n})$ holds as claimed. □

Next, we show that the total expected grammar size is $O(\delta \log \frac{n \log \sigma}{\delta \log n})$.

**Theorem 3.14.** *Consider the restricted block compression of a string $S \in [0 \mathinner{.\,.} \sigma)^n$ with measure $\delta$, where the functions $(\pi_k)_{k \geq 0}$ in Definition 3.5 are chosen uniformly at random. Then, the expected size of the resulting RLSLP is $O(\delta \log \frac{n \log \sigma}{\delta \log n})$.*

*Proof* Although Corollary 3.13 guarantees that $|\mathcal{S}| = O(\delta \log \frac{n \log \sigma}{\delta \log n})$, the remaining problem is that the size of the resulting grammar (i.e., the sum of production sizes) can be larger. Every symbol in $\Sigma \cup (\mathcal{A} \times \mathbb{Z}_{\geq 2})$ contributes $O(1)$ to the RLSLP size, so it remains to bound the total size of productions corresponding to symbols in $\mathcal{A}^{\geq 2}$. These symbols are introduced by restricted block parsing, i.e., they belong to $\mathcal{S}_{k+1} \setminus \mathcal{S}_k$ for odd $k \in [0 \mathinner{.\,.} \kappa]$. In order to estimate their contribution to the grammar size, we shall fix $\pi_0, \dots, \pi_k$ and compute the expectation with respect to the random choice of $\pi_{k+1}$. In this setting, we prove the following claim:

**Claim 3.15.** *Let $k > 0$ be odd and $T \in \mathcal{A}_k^m$ be a substring of $S_k$. Restricted block parsing $bc_{\pi_{k+1}, \mathcal{A}_{k+1}}(S_k)$ creates a block matching $T$ with probability $O(2^{-m})$.*

*Proof* Since $S_k = rle_{\mathcal{A}_k}(S_{k-1})$ and $\mathcal{A}_{k+1} = \mathcal{A}_k$, every two subsequent symbols of $T$ are distinct. Observe that if $T$ forms a block, then there is a value $t \in [1 \mathinner{.\,.} m]$ such that $\pi_{k+1}(T[1]) < \cdots < \pi_{k+1}(T[t]) > \cdots > \pi_{k+1}(T[m])$; otherwise, there would be a local minimum within every occurrence of $T$ in $S_{k-1}$. In particular, denoting $h := \lfloor m/2 \rfloor$, we must have $\pi_{k+1}(T[1]) < \cdots < \pi_{k+1}(T[h+1])$ (when $t > h$) or $\pi_{k+1}(T[m-h]) > \cdots > \pi_{k+1}(T[m])$ (when $t \leq h$). However, the probability that the values $\pi_{k+1}(\cdot)$ for $h+1$ consecutive characters form a strictly increasing (or strictly decreasing) sequence is at most $\frac{1}{(h+1)!}$: either exactly $\frac{1}{(h+1)!}$ (if the characters are distinct) or 0 (otherwise); this is because $\pi_{k+1}$ shuffles $\Sigma(S_k) \cap \mathcal{A}_{k+1}$ uniformly at random. Overall, we conclude that the probability that $T$ forms a block does not exceed $\frac{2}{(h+1)!} \leq 2^{-\Omega(m \log m)} \leq O(2^{-m})$. □

Next, note that every symbol in $\mathcal{S}_{k+1} \setminus \mathcal{S}_k$ is obtained by collapsing a block of $m$ active symbols created within $bc_{\pi_{k+1}, \mathcal{A}_{k+1}}(S_k)$ (with distinct symbols obtained from distinct blocks). By Lemma 3.10, the string $S_k$ has $O\left(m\left(1 + \frac{d_{r_{k,m}}(S)}{r_{k,m}}\right)\right)$ distinct substrings $T \in \mathcal{A}_{k+1}^m$ for $r_{k,m} := 2^{\lceil \log(50m\ell_{k+1}) \rceil}$. By Claim 3.15, any fixed substring

$T \in \mathcal{A}_{k+1}^m$ yields a symbol in $\mathcal{S}_{k+1} \setminus \mathcal{S}_k$ with probability $O(2^{-m})$. Consequently, the total contribution of symbols in $\mathcal{S}_{k+1} \setminus \mathcal{S}_k$ to the RLSLP size is, in expectation,

$$\sum_{m=2}^{\infty} O\left(\frac{m}{2^m}\left(1 + \frac{d_{r_{k,m}}(S)}{r_{k,m}}\right)\right) = O(1) + \sum_{m=2}^{\infty} O\left(\frac{m}{2^m} \cdot \frac{d_{r_{k,m}}(S)}{r_{k,m}}\right).$$

Across all the odd levels $k \in [0 \mathinner{..} \kappa)$, this sums up to at most

$$O(\kappa) + \sum_{k=0}^{\infty} \sum_{m=2}^{\infty} O\left(\frac{m}{2^m} \cdot \frac{d_{r_{k,m}}(S)}{r_{k,m}}\right)$$

$$= O(\log n) + \sum_{p=0}^{\infty} \sum_{m=2}^{\infty} \sum_{\substack{k \in \mathbb{Z}_{\geq 0} \\ r_{k,m} = 2^p}} O\left(\frac{m}{2^m} \cdot \frac{d_{2^p}(S)}{2^p}\right).$$

For every fixed $p, m \in \mathbb{Z}_{\geq 0}$, the number of integers $k \in \mathbb{Z}_{\geq 0}$ satisfying $r_{k,m} = 2^p$ is at most 6 (because $\ell_{k+6} = (4/3)^3 \ell_k > 2\ell_k$). Hence, for every $p \in \mathbb{Z}_{\geq 0}$, we have

$$\sum_{m=2}^{\infty} \sum_{\substack{k \in \mathbb{Z}_{\geq 0} \\ r_{k,m} = 2^p}} O\left(\frac{m}{2^m} \cdot \frac{d_{2^p}(S)}{2^p}\right) = \sum_{m=2}^{\infty} O\left(\frac{m}{2^m} \cdot \frac{d_{2^p}(S)}{2^p}\right) = O\left(\frac{d_{2^p}(S)}{2^p}\right).$$

Consequently, Fact 3.1 implies that the total expected grammar size is

$$O(\log n) + \sum_{p=0}^{\infty} O\left(\frac{d_{2^p}(S)}{2^p}\right) = O(\log n) + O(\delta \log \tfrac{n \log \sigma}{\delta \log n}) = O(\delta \log \tfrac{n \log \sigma}{\delta \log n}). \qquad \square$$

We are now ready to show how to build an RLSLP of size $O(\delta \log \frac{n \log \sigma}{\delta \log n})$ in linear expected time.

**Corollary 3.16.** *Given a string $S \in [0 \mathinner{..} \sigma)^n$ with measure $\delta$, we can build an RLSLP of size $O(\delta \log \frac{n \log \sigma}{\delta \log n})$ in $O(n)$ expected time.*

*Proof* We apply Definition 3.5 on top of the given string $S$, with functions $\pi_k$ chosen uniformly at random. It is an easy exercise to carry out this construction in $O(\sum_{k=0}^{\kappa} |S_k|) = O(n)$ worst-case time.

The expected size of the resulting RLSLP is at most $c \cdot \delta \log \frac{n \log \sigma}{\delta \log n}$ for some constant $c$; we can repeat the construction (with fresh randomness) until it yields an RLSLP of size at most $2c \cdot \delta \log \frac{n \log \sigma}{\delta \log n}$. By Markov's inequality, we succeed after $O(1)$ attempts in expectation. As a result, in $O(n)$ expected time, we obtain a grammar of total worst-case size $O(\delta \log \frac{n \log \sigma}{\delta \log n})$. $\qquad \square$

*Remark* 3.17 (Grammar height). In the algorithm of Corollary 3.16, we can terminate restricted block compression after $\lambda := 2\lfloor \log_{4/3} \frac{n}{\delta} \rfloor$ levels and complete the grammar with an initial symbol rule $A_\lambda \to S_\lambda[1] \cdots S_\lambda[|S_\lambda|]$ so that $\exp(A_\lambda) = S$. Corollary 3.9 yields $|S_\lambda| = O(1 + (\frac{3}{4})^{\lambda/2} n) = O(\delta)$, so the resulting RLSLP is still of size $O(\delta \log \frac{n \log \sigma}{\delta \log n})$; however, the height is now $O(\log \frac{n}{\delta})$.

# 4 Local Consistency Properties

We now show that the local consistency properties of our grammar enable fast indexed searches. Previous work [6] achieves this by showing that, thanks to the locally consistent parsing, only a set $M(P)$ of $O(\log |P|)$ pattern positions need to be analyzed for searching. To use this result, we now must take into account the pausing of symbols. Surprisingly, this modification allows for a much simpler definition of $M(P)$.

**Definition 4.1.** For every non-empty fragment $S[i \mathinner{.\,.} j]$ of $S$, we define

$$B_k(i,j) = \{p - i : p \in B_k \cap [i \mathinner{.\,.} j)\}$$

and

$$M(i,j) = \bigcup_{k \geq 0} \big(B_k(i,j) \setminus [2\alpha_{k+1} \mathinner{.\,.} j - i - \alpha_{k+1})$$
$$\cup \{\min(B_k(i,j) \cap [2\alpha_{k+1} \mathinner{.\,.} j - i - \alpha_{k+1}))\}\big),$$

where $\alpha_k = \lfloor 8\ell_k \rfloor$ and $\{\min \emptyset\} = \emptyset$.

Intuitively, the set $B_k(i,j)$ lists (the relative locations of) all level-$k$ phrase boundaries inside $S[i \mathinner{.\,.} j]$. For each level $k \geq 0$, we include in $M(i,j)$ the phrase boundaries that are close to either of the two endpoints of $S[i \mathinner{.\,.} j]$ (in the light of Lemma 3.7, it may depend on the context of $S[i \mathinner{.\,.} j]$ which of these phrase boundaries are preserved as level-$(k+1)$ boundaries) as well as the leftmost phrase boundary within the remaining internal part of $S[i \mathinner{.\,.} j]$.

**Lemma 4.2.** *The set $M(i,j)$ satisfies the following properties:*
1) *For each $k \geq 0$, if $B_k(i,j) \neq \emptyset$, then $\min B_k(i,j) \in M(i,j)$.*
2) *We have $|M(i,j)| = O(\log(j - i + 2))$.*
3) *If $S[i' \mathinner{.\,.} j'] = S[i \mathinner{.\,.} j]$, then $M(i',j') = M(i,j)$.*

*Proof* Let us express $M(i,j) = \bigcup_{k \geq 0} M_k(i,j)$, setting

$$M_k(i,j) := B_k(i,j) \setminus [2\alpha_{k+1} \mathinner{.\,.} j - i - \alpha_{k+1}) \cup \{\min(B_k(i,j) \cap [2\alpha_{k+1} \mathinner{.\,.} j - i - \alpha_{k+1}))\}.$$

As for Item 1, it is easy to see that $\min B_k(i,j) \in M_k(i,j)$: we consider two cases, depending on whether $\min B_k(i,j)$ belongs to $[2\alpha_{k+1} \mathinner{.\,.} j - i - \alpha_{k+1})$ or not.

As for Item 2, let us first argue that $|M_k(i,j)| = O(1)$ holds for every $k \geq 0$. Indeed, each element $q \in B_k(i,j) \cap [0 \mathinner{.\,.} 2\alpha_{k+1})$ corresponds to $q + i \in B_k \cap [i \mathinner{.\,.} i + 2\alpha_{k+1})$ and each element $q \in B_k(i,j) \cap [j - i - \alpha_{k+1} \mathinner{.\,.} j - i)$ corresponds to $q + i \in B_k \cap [j - \alpha_{k+1} \mathinner{.\,.} j)$. By Lemma 3.8, we conclude that $|M_k(i,j)| \leq 1 + (2 + \frac{8\alpha_{k+1}}{\ell_{k+1}}) + (2 + \frac{4\alpha_{k+1}}{\ell_{k+1}}) = O(1)$. Moreover, if $\ell_k > 4(j - i)$, then Lemma 3.8 further yields $|B_k(i,j)| = |B_k \cap [i \mathinner{.\,.} j)| \leq 1$. Since $M_k(i,j)$ and $B_{k+1}(i,j)$ are both subsets

of $B_k(i,j)$, this means that $\left|\bigcup_{k:\ell_k>4(j-i)} M_k(i,j)\right| \leq 1$. The number of levels $k$ satisfying $\ell_k \leq 4(j-i)$ is $O(\log(j-i+2))$, and thus
$$|M(i,j)| \leq O(1) \cdot O(\log(j-i+2)) + 1 = O(\log(j-i+2)).$$

As for Item 3, we shall prove by induction on $k$ that $M_k(i,j) \subseteq M(i',j')$. This implies $M(i,j) \subseteq M(i',j')$ and, by symmetry, $M(i,j) = M(i',j')$. In the base case of $k = 0$, we have
$$M_0(i,j) = ([0..2\alpha_1] \cup [j-i-\alpha_1..j-i)) \cap [0..j-i) = M_0(i',j').$$

Now, consider $k > 0$ and $q \in M_k(i,j)$. If $q \in B_k(i,j) \setminus [2\alpha_k..j-i-\alpha_k)$, then $q \in M_{k-1}(i,j)$, and thus $q \in M(i',j')$ holds by the inductive assumption. As for the remaining case, $M_k(i,j) \cap [2\alpha_k..j-i-\alpha_k] = M_k(i',j') \cap [2\alpha_k..j'-i'-\alpha_k]$ is a direct consequence of $B_k(i,j) \cap [2\alpha_k..j-i-\alpha_k] = B_k(i',j') \cap [2\alpha_k..j'-i'-\alpha_k]$, which follows from Lemma 3.7.                                                    □

**Definition 4.3.** Let $P$ be a substring of $S$ and let $S[i..j]$ be its arbitrary occurrence. We define $M(P) := M(i,j)$; by Item 3 of Lemma 4.2, this does not depend on the choice of the occurrence.

By Lemma 4.2, the set $M(P)$ is of size $O(\log|P|)$, yet, for every level $k \geq 0$ and every occurrence $P = S[i..j]$, it includes the leftmost phrase boundary in $B_k(i,j)$. Our index exploits the latter property for the largest $k$ with $B_k(i,j) \neq \emptyset$.

# 5 Indexing with our Grammar

In this section, we obtain our main result by adapting the results on attractors [6, Sec. 6] to our modified parsing.

**Definition 5.1** ([6])**.** The parse tree of an RLSLP is a rooted tree with leaves labeled with terminals and internal nodes labeled with nonterminals. The root is labeled with the initial symbol of the RLSP. If a node is labeled with a nonterminal $A$ associated with a production $A \to A_1 \cdots A_s$, then it has $s$ children labeled with $A_1, \ldots, A_s$, respectively, in the left-to-right order. If, instead, $A$ is associated with a production $A \to A_1^s$, then the node has $s$ children, each labeled with $A_1$.

**Definition 5.2** ([6])**.** The grammar tree of an RLSLP is obtained by pruning its parse tree: all but the leftmost occurrences of each nonterminal are converted into leaves and their subtrees are pruned. We treat rules $A \to A_1^s$ (assumed to be of size 2) as $A \to A_1 A_1^{[s-1]}$, where the node labeled $A_1^{[s-1]}$ is always a leaf ($A_1$ is also a leaf unless it is the leftmost occurrence of $A_1$).

Note that the grammar tree has exactly one internal node per distinct nonterminal and the total number of nodes is equal to the grammar size plus one. We associate each nonterminal $A$ with the only internal grammar tree node labeled with $A$. We also sometimes associate terminal symbols with grammar tree leaves.

The search algorithm classifies the occurrences of a pattern $P$ in $S$ into *primary* and *secondary*, according to the partition of $S$ induced by the grammar tree leaves.

**Definition 5.3** ([6])**.** The leaves of the grammar tree induce a partition of $S$ into phrases. An occurrence $S[t \mathinner{.\,.} t + m)$ of $P[1 \mathinner{.\,.} m]$ is *primary* if the lowest grammar tree node deriving a fragment of $S$ that contains $S[t \mathinner{.\,.} t + m)$ is internal (or, equivalently, the occurrence crosses the boundary between two phrases); otherwise, the occurrence is *secondary*.

The general idea of the search is to find the primary occurrences by looking for prefix-suffix partitions of $P$ and then derive the secondary occurrences from the primary ones [10].

## 5.1 Finding the primary occurrences

Let a nonterminal $A$ be the lowest (internal) grammar tree node that covers a primary occurrence $S[t \mathinner{.\,.} t + m)$ of $P[1 \mathinner{.\,.} m]$. Then, if $A \to A_1 \cdots A_s$, there exist $i \in [1 \mathinner{.\,.} s]$ and $q \in [1 \mathinner{.\,.} m]$ such that a suffix of $\exp(A_i)$ matches $P[1 \mathinner{.\,.} q]$ whereas a prefix of $\exp(A_{i+1}) \cdots \exp(A_s)$ matches $P[q \mathinner{.\,.} m]$. The idea is to index all the pairs $(\exp(A_i)^{rev}, \exp(A_{i+1}) \cdots \exp(A_s))$ and find those where the first and second component are prefixed by $(P[1 \mathinner{.\,.} q])^{rev}$ and $P[q \mathinner{.\,.} m]$, respectively. Note that there is exactly one such pair per border between two consecutive phrases (or leaves in the grammar tree). On the other hand, if $A \to A_1^s$, then there exists $q \in [1 \mathinner{.\,.} m]$ such that a suffix of $\exp(A_1)$ matches $P[1 \mathinner{.\,.} q]$ whereas a prefix of $\exp(A_1)^{s-1}$ matches $P[q \mathinner{.\,.} m]$. We will also index the pairs $(\exp(A_1)^{rev}, \exp(A_1)^{s-1})$ and find those where the first and second component are prefixed by $(P[1 \mathinner{.\,.} q])^{rev}$ and $P[q \mathinner{.\,.} m]$, respectively.

**Definition 5.4** ([6])**.** Let $v$ be the lowest (internal) grammar tree node that covers a primary occurrence $S[t \mathinner{.\,.} t + m)$ of $P$. Let $v_i$ be the leftmost child of $v$ that overlaps $S[t \mathinner{.\,.} t + m)$. We say that the node $v$ is the *parent* of the primary occurrence $S[t \mathinner{.\,.} t + m)$ of $P$ and the node $v_i$ is its *locus*.

The index of [6] builds a two-dimensional grid data structure [21]. First, two multisets of strings, $\mathcal{X}$ and $\mathcal{Y}$, are constructed. For every nonterminal $A \to A_1 \cdots A_s$ and every $i \in [1 \mathinner{.\,.} s)$, the strings $\exp(A_i)^{rev}$ and $\exp(A_{i+1}) \cdots \exp(A_s)$ are added to $\mathcal{X}$ and $\mathcal{Y}$, respectively. For every nonterminal $A \to A_1^s$, the strings $\exp(A_1)^{rev}$ and $\exp(A_1)^{s-1}$ are added to $\mathcal{X}$ and $\mathcal{Y}$, respectively. For each nonterminal $A \to A_1 \cdots A_s$ and each $i \in [1 \mathinner{.\,.} s)$, the grid contains a point $(x, y)$ such that $x$ is the lexicographic rank of $\exp(A_i)^{rev}$ in $\mathcal{X}$ and $y$ is the lexicographic rank of $\exp(A_{i+1}) \cdots \exp(A_s)$ in $\mathcal{Y}$; the point is associated with the locus $A_i$. For each nonterminal $A \to A_1^s$, the grid contains a point $(x, y)$ such that $x$ is the lexicographic rank of $\exp(A_1)^{rev}$ in $\mathcal{X}$ and $y$ is the lexicographic rank of $\exp(A_1)^{s-1}$ in $\mathcal{Y}$; the point associated with the locus $A_1$.

Note that the number of strings in $\mathcal{X}$ and $\mathcal{Y}$, as well as the number of points in the grid, are bounded by the grammar size, $g = O(\delta \log \frac{n \log \sigma}{\delta \log n})$ in our case.

Given a partition $P = P[1 \mathinner{.\,.} q] \cdot P(q \mathinner{.\,.} m]$ to look for primary occurrences, the query algorithm searches $\mathcal{X}$ for the lexicographic range $[x_1, x_2]$ of all the strings that are prefixed with $P[1 \mathinner{.\,.} q]^{rev}$, and $\mathcal{Y}$ for the lexicographic range $[y_1, y_2]$ of all the strings that are prefixed with $P(q \mathinner{.\,.} m]$. By interpreting the lexicographic ranges as $x$-coordinate and $y$-coordinate ranges, respectively, it follows that the grid points within the orthogonal range $[x_1, x_2] \times [y_1, y_2]$ corresponds precisely to all the $occ_q$ primary occurrences of $P$ cut at position $q$. The geometric data structure, using $O(g)$ space, finds the points in such range of the grid in time $O((occ_q + 1) \log^\epsilon g)$, where $\epsilon > 0$ is an arbitrary constant fixed at construction time [21, Thm. 2.1, using $B = \log^\epsilon n / \log \log n$].

In order to efficiently search $\mathcal{X}$ for all the reversed prefixes $P[1 \mathinner{.\,.} q]^{rev}$ and $\mathcal{Y}$ for all the suffixes $P(q \mathinner{.\,.} m]$, we make use of the following lemma.

**Lemma 5.5** ([6, Lem. 6.5]). *Let $\mathcal{S}$ be a set of strings augmented with a data structure supporting:*

- *extraction, in time $f_e(\ell)$, of any length-$\ell$ prefix of strings in $\mathcal{S}$, and*
- *computation, in time $f_h(\ell)$, of the Karp–Rabin signature $\phi$ of any length-$\ell$ prefix of strings in $\mathcal{S}$.*

*We can then build a data structure of $O(|\mathcal{S}|)$ words such that, later, we can solve the following problem in $O(m + \tau(f_h(m) + \log m) + f_e(m))$ time: given a pattern $P[1 \mathinner{.\,.} m]$ and $\tau > 0$ suffixes $Q_1, \ldots, Q_\tau$ of $P$, find the ranges of strings in (the lexicographically-sorted) $\mathcal{S}$ prefixed by $Q_1, \ldots, Q_\tau$.*

We use this lemma for $\mathcal{S} = \mathcal{X}$ and $\mathcal{S} = \mathcal{Y}$, so the extraction of a prefix of a string in $\mathcal{S}$ corresponds to extracting the suffix of $\exp(A_i)$ for some nonterminal $A_i$ (in the case $\mathcal{S} = \mathcal{X}$) or to extracting a prefix of $\exp(A_{i+1}) \cdots \exp(A_s)$ or of $\exp(A_1)^{s-1}$ (in the case $\mathcal{S} = \mathcal{Y}$). In this second case, we need to extract the complete expansion of zero or more nonterminals, and then a prefix of the expansion of a nonterminal. It is shown in [6] that prefixes and suffixes of the expansions of nonterminals can be extracted in linear time, which yields $f_e(\ell) = O(\ell)$ in our grammar.

**Lemma 5.6** ([6, Lem. 6.6]). *For every RLSLP of size $g$, there exists a data structure of size $O(g)$ such that, for every nonterminal $A$, any prefix or suffix of $\exp(A)$ can be obtained in time proportional to its length.*

The second requirement of Lemma 5.5 is to compute the Karp–Rabin [22] signature $\phi$ on prefixes of strings in $\mathcal{S}$. Those are functions of the form $\phi(S) = \left( \sum_{i=1}^{|S|} S[i] \cdot b^{|S|-i} \right) \bmod p$, for some prime number $p$ and a randomly chosen integer $b \in [1 \mathinner{.\,.} p)$. To prove Lemma 5.5, the signature $\phi$ is chosen in [6] so that there is no collision between substrings of $\mathcal{S}$, and thus the searches for the suffixes $Q_i$ are deterministically correct. In a followup to the conference

version of our paper, Navarro [23] showed how to obtain $f_h(\ell) = O(\log \ell)$ in our grammar.

**Lemma 5.7** ([23, Lem. 12])**.** *The RLSLP of Section 3 can be enhanced with $O(g)$ additional space so that the Karp–Rabin signature of any length-$\ell$ prefix of a string in $\mathcal{X}$ or $\mathcal{Y}$ can be extracted in time $f_h(\ell) = O(\log \ell)$.*

With this result, we obtain time $O(m + \tau \log m)$ to perform all the string searches using Lemma 5.5. Overall, if we have identified $\tau$ cuts of $P$ that suffice to find all of its primary occurrences in $S$, then we can find all the $occ_p \leq occ$ primary occurrences of $P$ in time $O(m + \tau \log m + (occ_p + \tau) \log^\epsilon g)$.

## 5.2 Parsing the pattern

The next step is to set a bound for $\tau$ with our parsing and show how to find the corresponding cuts. As shown below, since only the leftmost and rightmost $O(1)$ phrase boundaries in each level of the parsing of $P$ can differ from those in the parsing of any occurrence of $P$ in $T$, the leftmost phrase boundary within every primary occurrence of $P$ is guaranteed to belong to $M(P)$.

**Lemma 5.8.** *Using our grammar of Section 3, there are only $\tau = O(\log m)$ cuts $P = P[1 \mathinner{.\,.} q] \cdot P[q \mathinner{.\,.} m]$ yielding primary occurrences of $P[1 \mathinner{.\,.} m]$. These positions $q$ belong to $M(P) + 1 = \{r + 1 : r \in M(P)\}$ (see Definition 4.3).*

*Proof* Let $A$ be the parent of a primary occurrence $S[t \mathinner{.\,.} t + m)$, and let $k$ be the round where $A$ is formed. There are two possibilities:

(1) $A \to A_1 \cdots A_s$ is a block-forming rule, and a suffix of $\mathtt{exp}(A_i)$ matches $P[1 \mathinner{.\,.} q]$, for some $i \in [1 \mathinner{.\,.} s)$ and $q \in [1 \mathinner{.\,.} m)$. This means that $q - 1 = \min B_{k-1}(t, t + m - 1)$.

(2) $A \to A_1^s$ is a run-length nonterminal and a suffix of $\mathtt{exp}(A_1)$ matches $P[1 \mathinner{.\,.} q]$, for some $q \in [1 \mathinner{.\,.} m)$. This means that $q - 1 = \min B_{k-1}(t, t + m - 1)$.

In either case, $q \in M(P) + 1$ by Lemma 4.2. Further, $|M(P)| = O(\log m)$. $\square$

The parsing of $P$ is performed done in $O(m)$ time much as in previous work [6, Sec. 6.1], with the difference that we have to care about paused symbols. Essentially, we store the permutations $\pi_k$ drawn when indexing $S$ and use them to parse $P$ in the same way, round by round, aiming to create the same symbols. Actually, we parse the string $P^* = \#P\$$, where $\#$ and $\$$ are two symbols that appear neither in $S$ nor in $P$. We use the occurrence of $P$ in $P^*$ to build $M(P)$.

As observed in the proof of Corollary 3.13, we have $\sum_{k=0}^\kappa |\mathcal{S}_k \cap \mathcal{A}_{k+1}| = O(\delta \log \frac{n \log \sigma}{\delta \log n})$. Hence, we store the values of $\pi_{k+1}$ only for the active symbols in $S_k$ for every $k \in [0 \mathinner{.\,.} \kappa]$; the values $\pi_{k+1}$ for the remaining symbols do not affect the placement of block boundaries in Definition 3.4: If $S_k[j], S_k[j+1] \in \mathcal{A}_{k+1}$, then, due to the condition imposed on $\pi_{k+1}$ in Definition 3.5, $j$ may

only be a local minimum if $S_k[j-1] \in \mathcal{A}_{k+1}$. When parsing $P$, we can simply assume that $\pi_{k+1}(A) = 0$ on the paused symbols $A \in \mathcal{S}_k \setminus \mathcal{A}_{k+1}$ and obtain the same parsing of $S$. By storing the values of $\pi_k$ only for the active symbols, we use $O(\delta \log \frac{n \log \sigma}{\delta \log n})$ total space.

In order to find the correct nonterminals among those used in the parsing of $S$, we store two perfect hash tables [24]. In the first one, we keep all the keys $(A_1, s)$, with associated information $A$, corresponding to the grammar rules of the form $A \rightarrow A_1^s$. In the second one, we keep, for each rule $A \rightarrow A_1 \cdots A_s$ formed in $S$, the key $(A_1 \cdots A_s)$ with associated information $A$. The size of the hash tables is proportional to the size $g$ of the grammar.

Let us describe the first two rounds of the parsing [6]. We first traverse $P^* = P_0^*$ left-to-right and identify the runs $a^\ell$ in $O(m)$ time. For each such run, we search the first hash table for the key $(a, \ell)$, finding its corresponding nonterminal $A$, with which we replace the run (see below for the case where the run does not appear in the hash table). The result of this pass is a new sequence $P_1^*$. We then traverse $P_1^*$, finding the local minima according to $\pi_1$, and thus identifying the blocks. To do this in $O(m)$ time as well, we search the second hash table for the identified blocks, replacing them by the corresponding nonterminals, and forming the sequence $P_2^*$.

By Corollary 3.9, the number of phrases in round $k$ is less than $1 + \frac{4m}{\ell_{k+1}}$, which gives us at most $h = 12 + 2\lfloor \log_{4/3} m \rfloor = O(\log m)$ parsing rounds and a total of $\sum_{k=0}^{h}(1 + \frac{4m}{\ell_{k+1}}) = O(m)$ symbols processed along the parsing of $P$. Since we spend constant time per symbol along the parse the whole parsing takes time $O(m)$. Construction of the set $M(P)$ from Definitions 4.1 and 4.3 (along with the auxiliary sets $B_k(i,j)$ and $M(i,j)$) also takes $O(m)$ time.

Note that $P_k^*$ may contain runs and blocks that do not occur in $S_k$. By Lemma 3.7, other than those overlapping the first $2\alpha$ or the last $\alpha$ positions of of $P$, where $\alpha := \lfloor 8\ell_k \rfloor$, any run or block formed inside $P^*$ must also be formed inside every occurrence of $P$ in $S$. Consequently, if a run or a block does not overlap those extreme positions, yet it does not appear in the hash table, we can abandon the search because $P$ cannot occur in $S$. On the other hand, by Lemma 3.8, there can be only $O(1)$ symbols overlapping those areas in each level $P_k^*$, and thus $O(\log m)$ in total. We can then gather those unknown runs and blocks appearing in the extremes of $P^*$ in order to consistently assign them new nonterminals and arbitrary unused $\pi_k$ values. We then proceed normally with subsequent levels of the parsing. Traversing the set linearly in order to detect if they reappear in the parsing adds up to just $O(\log^2 m)$ total time.

## 5.3 Secondary occurrences and short patterns

Thus far, we have obtained the loci of the primary occurrences, but not yet their positions in $S$. Further, we must find the secondary occurrences that derive from the primary ones. In order to find those, we use a technique that works for any arbitrary RLSLP and within $O(g)$ space [6, Sec. 6.4]. For each grammar tree node $v$ labeled $A$, we store (i) $u = v.anc$, the nearest ancestor

of $v$, labeled $B$, such that $u$ is the root or $B$ labels more than one node in the grammar tree; (ii) $v.\textit{offs}$, the offset at which $v$ (i.e., $\mathsf{exp}(A)$) starts inside $u$ (i.e., $\mathsf{exp}(B)$); and (iii) $v.\textit{next}$, the next node in preorder traversal of the grammar tree labeled $A$, or *null* if $v$ is the last node labeled $A$. The first two fields are undefined for the nodes labeled $A^{[s-1]}$ that we create in the grammar tree (these nodes do not exist in the parse tree), whereas these nodes participate in the $v.\textit{next}$ list of label $A$.

For each primary occurrence found with the partition $P = P[1..q] \cdot P[q..m]$, the point we found in the grid gives us its locus $v$ in the grammar tree. Say $v$ is labeled $A_i$, and its parent $u$ is labeled $A$, with $A \to A_1 \cdots A_s$. For each such node $v$, we store $|\mathsf{exp}(A_1 \cdots A_i)|$, so the offset of $P$ within $\mathsf{exp}(A)$ is $o := |\mathsf{exp}(A_1 \cdots A_i)| - q + 1$. To find the offset within $S$ of the primary occurrence, we repeatedly traverse pointers from $u$ to $u.\textit{anc}$ and update $o := o + u.\textit{offs}$, until reaching the root. At this point, we can report an occurrence $S[o..o+m)$ of $P$.

In our upwards way to the root, we will also report secondary occurrences. For every node $u$ we visit in an upward path, we recursively continue not only by $u.\textit{anc}$ but also by $u.\textit{next}$; the offset $o$ is retained in this second case. The tree of recursive calls is binary and reports a different occurrence of $P$ in $S$ at each leaf reaching the initial symbol. Therefore, the positions of all the occurrences, primary and secondary, are reported in constant amortized time each.[3]

The case where $i = 1$ and the parent $u$ of $v$ is labeled $A \to A_1^s$ is special. We compute $o := |\mathsf{exp}(A_1)| - q + 1$ and report occurrences in $u$ with offsets $o+i \cdot |\mathsf{exp}(A_1)|$ for $i = 0, 1, \ldots$ as long as $o+i \cdot |\mathsf{exp}(A_1)|+m-1 \le s \cdot |\mathsf{exp}(A_1)|$, apart from the call to $v.\textit{next}.\textit{next}$. On the other hand, the nodes $u$ labeled $A_1^{[s-1]}$ reached by pointers *next* along the recursive calls are treated exactly as $s-1$ copies of $A_1$, triggering $s-1$ calls to $u.\textit{anc}$ with offsets $o + i \cdot |\mathsf{exp}(A_1)|$ for $i \in [1..s)$, and also one call to $u.\textit{next}$ with offset $o$.

Plugged with the preceding results, the total space of our index is $O(\delta \log \frac{n \log \sigma}{\delta \log n})$ and its search time is $O(m + \tau(\log^\epsilon g + \log m) + occ \log^\epsilon g) = O(m + (\log m + occ) \log^\epsilon g)$. This bound exceeds $O(m + (occ + 1) \log^\epsilon g)$ only when $m = O(\log^\epsilon g \log \log g)$. In that case, however, $O(\log m \log^\epsilon g) = O(\log \log g \cdot \log^\epsilon g)$, which becomes $O(\log^\epsilon g)$ again if we infinitesimally adjust $\epsilon$. Our complexity then becomes $O(m + \log^\epsilon g + occ \log^\epsilon g)$.

The final touch is to reduce that complexity to $O(m + \log^\epsilon \delta + occ \log^\epsilon g)$. This is relevant only when $occ = 0$, so we need a way to detect in time $O(m + \log^\epsilon \delta)$ that $P$ does not occur in $S$. We already do this in time $O(m + \log^\epsilon g)$ by parsing $P$ and searching for its cuts in the geometric data structure. To reduce the time, we note that $\log^\epsilon g \le \log^\epsilon \delta + \log^\epsilon \frac{g}{\delta}$, so it suffices to detect in $O(m)$ time the patterns of length $m \le \ell := \log^\epsilon \frac{g}{\delta}$ that do not occur in $S$. By definition of $\delta$, there are at most $\delta \ell^2$ substrings of length at most $\ell$ in $S$, so we can store them all in an (uncompressed) trie using total

---

[3]The unproductive tests $u.\textit{next} = \textit{null}$ are charged to the primary occurrence $v$ if the label of $u$ is $A$, or to the first secondary occurrence of $u$, which exists by the definition of $u.\textit{anc}$, otherwise.

space $O(\delta\ell^2) = O(\delta \log^{2\epsilon} \frac{g}{\delta}) = O(g) = O(\delta \log \frac{n \log \sigma}{\delta \log n})$. By implementing the trie children with perfect hashing [24], we can verify in $O(m)$ time whether a pattern of length $m \le \ell$ occurs in $S$. We then obtain Theorem 1.1.

## 5.4 Optimal search time

We now show how to obtain optimal search time, $O(m + occ)$, by slightly increasing the size of our $\delta$-bounded index. As in previous work [6, Sec. 6.7], we use a larger geometric structure, of size $O(g \log^\epsilon g)$ for some constant $\epsilon > 0$, which reports in $O(\log \log g)$ time per query and $O(1)$ per result. This structure is constructed in $O(g \log g)$ expected time using $O(g \log^\epsilon g)$ working space [25, Thm. 2].

With the enhanced geometric structure, the query time automatically drops to $O(m + \log m \log \log g + occ)$. This can be written as $O(m + \ell + occ)$, with $\ell = \log \log g \log \log \log g$ [6, Sec. 6.7], because the middle term dominates only if $m \le \log m \log \log g$.

In order to achieve the optimal search time, we need only to care about patterns of length up to $\ell$ that occur less than $\ell$ times, since otherwise the term $occ$ absorbs $\ell$. To do this, we store all the text substrings of length up to $\ell$ in an uncompressed trie $C$. This trie stores the number of times each of its nodes occurs in the text, and stores in its leaves at depth $\ell$ the actual list of those occurrences, in case there are at most $\ell$. By definition of $\delta$, there are at most $\delta\ell$ distinct substrings of length $\ell$, so the trie contains at most $\delta\ell^2$ nodes. Moreover, across all the leaves, we store at most $\delta\ell^2$ occurrences. The total space for $C$ is then $O(\delta\ell^2) \subseteq O(\delta \log^\epsilon g)$ for any constant $\epsilon > 0$.

To search for a pattern $P$ of length up to $\ell$, we first search for it in $C$ and verify if its trie node indicates that it occurs more than $\ell$ times. If so, we use the normal search, as described above, which on the enhanced grid takes time $O(m + \ell + occ) = O(m + occ)$. Otherwise, its occurrences are collected from all the leaves descending from its trie node, also in time $O(m + occ)$. Instead, if the pattern length exceeds $\ell$, it is also searched normally, in time $O(m + \ell + occ) = O(m + occ)$.

To build $C$, we slide a window of length $\ell$ through the whole text $S$, maintaining a Karp–Rabin fingerprint $\phi$ for the current window. We store the distinct signatures $\phi$ found in a hash table $H$, with a counter of its corresponding length-$\ell$ substring in $S$. When a new $\phi$ value appears, the underlying string is inserted in $C$, a pointer from $H$ is set to point to the corresponding trie leaf, and the list of the occurrences of the substring is initialized in the corresponding leaf, with its first position just found. Every new occurrence found is added in its trie leaf, until the length of its list exceeds $\ell$, in which case the list is deleted and not maintained anymore.

Per the Karp–Rabin formula given above, it takes $O(1)$ time to compute $\phi(S \cdot b)$ from $\phi(a \cdot S)$; thus, it takes $O(n)$ time to compute the fingerprints for all the length-$\ell$ windows of $S$. It takes $O(n)$ expected time to maintain $H$ (also considering collisions) and collect the occurrences, plus $O(\delta\ell^2)$ time to insert strings in $C$. We can propagate the number of occurrences of each leaf upwards

in the trie, in $O(\delta\ell)$ additional time. Added to the $O(g \log g)$ expected time to build the enhanced grid, the expected construction time is $O(n+g\log g) \subseteq O(n + \delta\log\frac{n\log\sigma}{\delta\log n}\log n)$. This completes the proof of Theorem 1.2.

# 6 Counting pattern occurrences

In this section, we show how to efficiently count pattern occurrences within $\delta$-bounded space by adapting previous results [6, Sec. 7] to our parsing.

## 6.1 Near-optimal counting time

Christiansen et al. [6] build an RLSLP of size $O(\gamma\log\frac{n}{\gamma})$ on $S[1 .. n]$ to count how many times a pattern $P[1 .. m]$ occurs in $S$ in the near-optimal time $O(m + \log^{2+\epsilon} n)$, for any constant $\epsilon > 0$. The idea is that, instead of tracking all the secondary occurrences that are reached from each point of the grid that lies in the appropriate ranges (recall Section 5.3), the number of those occurrences is stored with the corresponding grid point, so they just need to sum up those numbers. The only problem is with the run-length rules, as in those cases the number of occurrences is not a function of the points only. This case is handled using properties of their particular parsing.

Except for the problem of the run-length rules, the idea works on any grammar. As seen in Section 5, using our grammar of size $g = O(\delta\log\frac{n\log\sigma}{\delta\log n})$, there are $O(\log m)$ cutting positions that must be tried when searching for $P$. We need $O(m)$ time to identify the $O(\log m)$ relevant two-dimensional ranges, and then count the number of occurrences in each such range in time $O(\log^{2+\epsilon} g)$, using the same geometric data structure as in previous work [6]. Our counting time is then $O(m + \log m\log^{2+\epsilon} g)$. If $m \le \log m\log^{2+\epsilon} g$, then we have $\log m\log^{2+\epsilon} g \in O(\log^{2+\epsilon} g\log\log g)$. In that case, infinitesimally adjusting $\epsilon$, we have $\log m\log^{2+\epsilon} g \in O(\log^{2+\epsilon} g)$; thus, our counting time can be written as $O(m + \log^{2+\epsilon} g)$.

All that is needed is then to handle the problem of the run-length rules in our particular grammar. Just as in previous work [6], we will rely on a property of the shortest period of the expansion of the string generated by a run-length rule. In our grammar, we must take care of the paused symbols.

**Definition 6.1.** A string $P[1 .. m]$ has a period $p$ if $P$ consists of $\lfloor m/p \rfloor$ consecutive copies of $P[1 .. p]$ plus a (possible empty) prefix of $P[1 .. p]$.

The only property required for the mechanism designed for run-length rules [6] is the one we prove in the following lemma.

**Lemma 6.2** (cf. [6, Lemma 7.2]). *For every run-length rule $A \to A_1^s$ in our grammar, the shortest period of $\exp(A)$ is $|A_1| := |\exp(A_1)|$.*

*Proof* Let $\exp(A) = S(i \mathinner{\ldotp\ldotp} j]$. Since $\exp(A)$ consists of $s$ consecutive copies of $\exp(A_1)$, then $|A_1|$ is a period of $\exp(A)$. From the Periodicity Lemma [26], we know that the shortest period $p$ of $\exp(A)$ satisfies $p = \gcd(p, |A_1|)$; thus, $d := |A_1|/p$ is an integer. Let $v$ be the node labeled $A$ in the parse tree of the grammar, and let $r+1$ denote the level of the run represented by $v$, so $A$ is a symbol in $S_{r+1}$ and $A_1$ is a symbol in $S_r$.

**Claim 6.3.** *For every $k \leq r$, there are $ds$ identical fragments in $S_k$ that expand to the (identical) fragments $S(i \mathinner{\ldotp\ldotp} i+p], S(i+p \mathinner{\ldotp\ldotp} i+2p], \ldots, S(j-p \mathinner{\ldotp\ldotp} j]$.*

*Proof* Since $p$ is a period of $S(i \mathinner{\ldotp\ldotp} j] = S_0(i \mathinner{\ldotp\ldotp} j]$, we have $S_0(i \mathinner{\ldotp\ldotp} i+p] = \cdots = S_0(j-p \mathinner{\ldotp\ldotp} j]$. Hence, the claim holds trivially for $k = 0$. For $k > 0$, the inductive assumption yields $ds$ identical fragments $S_{k-1}(i_k \mathinner{\ldotp\ldotp} i_k + p_k], \ldots, S_{k-1}(j_k - p_k \mathinner{\ldotp\ldotp} j_k]$ that expand to $S(i \mathinner{\ldotp\ldotp} i+p], \ldots, S(j-p \mathinner{\ldotp\ldotp} j]$, respectively. Since $i, i + dp, j$ are all level-$r$ phrase boundaries, they are also level-$k$ phrase boundaries. Consequently, there is a block boundary after $S_{k-1}[i_k]$, $S_{k-1}[i_k + dp_k]$, and $S_{k-1}[j_k]$. In particular, $S_{k-1}(i_k \mathinner{\ldotp\ldotp} j_k]$ is parsed into full blocks, which are collapsed to $S_k(i_{k+1} \mathinner{\ldotp\ldotp} j_{k+1}]$. We shall prove that, for every $\delta \in [0 \mathinner{\ldotp\ldotp} p_k)$ and $t \in [0 \mathinner{\ldotp\ldotp} ds)$, there is a block boundary after $S_{k-1}[i_k + tp_k + \delta]$ if and only if there is one after $S_{k-1}[i_k + dp_k + \delta]$. For this, we consider several cases.

- If $tp_k + \delta = 0$, there are block boundaries after both $S_{k-1}[i_k]$ and $S_{k-1}[i_k + dp_k]$.

- If $k$ is odd and $tp_k + \delta \geq 1$, then
  $$S_{k-1}[i_k + tp_k + \delta \mathinner{\ldotp\ldotp} i_k + tp_k + \delta + 1] = S_{k-1}[i_k + dp_k + \delta \mathinner{\ldotp\ldotp} i_k + dp_k + \delta + 1].$$
  Since run-length encoding places a block boundary after $S_{k-1}[\ell]$ solely based on $S_{k-1}[\ell \mathinner{\ldotp\ldotp} \ell+1]$, we conclude that there is a block boundary after $S_{k-1}[i_k + tp_k + \delta]$ if and only if there is one after $S_{k-1}[i_k + dp_k + \delta]$.

- If $k$ is even, $tp_k + \delta \geq 2$, and $p_k \geq 2$, then
  $$S_{k-1}[i_k + tp_k + \delta - 1 \mathinner{\ldotp\ldotp} i_k + tp_k + \delta + 1] = S_{k-1}[i_k + dp_k + \delta - 1 \mathinner{\ldotp\ldotp} i_k + dp_k + \delta + 1].$$
  Since the restricted block parsing places a block boundary after $S_{k-1}[\ell]$ solely based on $S_{k-1}[\ell - 1 \mathinner{\ldotp\ldotp} \ell + 1]$, we conclude that there is a block boundary after $S_{k-1}[i_k + tp_k + \delta]$ if and only if there is one after $S_{k-1}[i_k + dp_k + \delta]$.

- If $k$ is even, $tp_k + \delta = 1$, and $p_k \geq 2$, then we need to consider several possibilities. If either $S_{k-1}[i_k + 1] \notin \mathcal{A}_k$ or $S_{k-1}[i_k + 2] \notin \mathcal{A}_k$, then, due to $S_{k-1}[i_k + 1 \mathinner{\ldotp\ldotp} i_k + 2] = S_{k-1}[i_k + dp_k + 1 \mathinner{\ldotp\ldotp} i_k + dp_k + 2]$, restricted block parsing places block boundaries both after $S_{k-1}[i_k + 1]$ and $S_{k-1}[i_k + dp_k + 1]$. Thus, we can assume that both $S_{k-1}[i_k + 1] \in \mathcal{A}_k$ and $S_{k-1}[i_k + 2] \in \mathcal{A}_k$. Since there is a block boundary after $S_{k-1}[i_k]$, then $i_k = 0$ or $\pi_k(S_{k-1}[i_k]) < \pi_k(S_{k-1}[i_k + 1])$; the latter holds even if $S_{k-1}[i_k] \notin \mathcal{A}_k$ because $\pi_k$ assigns larger values to symbols in $\mathcal{A}_k$ than to symbols outside $\mathcal{A}_k$ (see Definition 3.5). In either case, this means that $\pi_k(S_{k-1}[i_k + 1])$ is not a local minimum, so restricted block parsing does not place a block boundary after $S_{k-1}[i_k + 1]$. Similarly, since there is a block boundary after $S_{k-1}[i_k + dp_k]$, then $\pi_k(S_{k-1}[i_k + dp_k]) < \pi_k(S_{k-1}[i_k + dp_k + 1])$, so there is no block boundary after $S_{k-1}[i_k + dp_k + 1]$.

- If $k$ is even and $p_k = 1$, then all the symbols of $S_{k-1}(i_k \mathinner{\ldotp\ldotp} j_k]$ are equal. The run they constitute has not collapsed at level $k - 1$, so the underlying symbol does not belong to $\mathcal{A}_{k-1} = \mathcal{A}_k$. Consequently, all symbols of $S_{k-1}(i_k \mathinner{\ldotp\ldotp} j_k]$ form length-1 blocks. In particular, there are block boundaries after both $S_{k-1}[i_k + t]$ and $S_{k-1}[i_k + d]$.

All in all, we have proved that all fragments $S_{k-1}(i_k + tp_k \ldots i_k + (t+1)p_k]$ consist of full blocks and that they are parsed identically. Hence, each of them gets collapsed to an identical fragment of $S_k$ that expands to $S(i + tp \ldots i + (t+1)p]$. □

Applying Claim 6.3 to $k = r$, we conclude that $i, i+p, i+2p, \ldots, j$ are all level-$r$ phrase boundaries. In particular, $S(i \ldots j]$ consists of at least $ds$ level-$r$ phrases. However, $S(i \ldots j]$ constitutes exactly one level-$(r+1)$ phrase corresponding to $A$. Since $A \to A_1^s$, we conclude that $ds = s$, i.e., $p = |A_1|$ holds as claimed. □

By Lemma 6.2, we can safely use the original method [6] to handle the run-length rules. Thus, using $O(\delta \log \frac{n \log \sigma}{\delta \log n})$ space, we can build in $O(n \log n)$ expected time a data structure that can count the occurrences of a pattern $P[1 \ldots m]$ within $S[1 \ldots n]$ in near-optimal time $O(m + \log^{2+\epsilon} n)$. This proves the first part of Theorem 1.3.

## 6.2 Optimal counting time

We now show how to reduce the counting time offered by our index to the optimal $O(m)$, at the cost of increasing the space consumption by a factor of $O(\log n)$. If $\log g \geq \frac{n \log \sigma}{\delta \log n}$, then we can simply use a compact index [27], which takes $O(\frac{n \log \sigma}{\log n}) \subseteq O(\delta \log g) = O(\delta \log(\delta \log \frac{n \log \sigma}{\delta \log n}))$ space, can be constructed in $O(n)$ expected time, and answers counting queries in $O(m)$ time. Thus, we henceforth assume that $\log g < \frac{n \log \sigma}{\delta \log n}$.

In the same way as done in previous work [6, Sec. 7.1], we use a larger geometric structure that speeds up counting time. This structure [25, Thm. 3] supports orthogonal range counting queries in $O(\log g)$ time using $O(g \log g)$ space, and is built in $O(g \log^2 g)$ expected time. The counting time of our index then becomes $O(m + \log m \log g)$.

Using a compact trie similar to the one described in Section 5.4, we index all the substrings of length at most $\ell = \lceil \log g \log \log g \rceil$. The only difference with the trie we used before is that the trie we use now is compact (i.e., internal nodes with one child are not stored explicitly) and that we do not store the list of occurrences but just the number of occurrences at each node. Since there are $\delta \ell$ substrings of length exactly $\ell$, the space consumption of the compact trie is $O(\delta \ell) = O(\delta \log g \log \log g)$.

Our counting time $O(m + \log m \log g)$ is in $O(m)$ when $m > \ell$ because then $\log g = O(\frac{\ell}{\log \ell}) \subseteq O(\frac{m}{\log m})$. If we answer queries of length up to $\ell$ using the compact trie and any other query using the geometric structure, our counting time is always $O(m)$.

The space consumption of our data structure is then $O(g \log g + \delta \log g \log \log g)$. The first term is $O(\delta \log \frac{n \log \sigma}{\delta \log n} \log(\delta \log \frac{n \log \sigma}{\delta \log n}))$ because $g = O(\delta \log \frac{n \log \sigma}{\delta \log n})$. Due to our assumption $\log g < \frac{n \log \sigma}{\delta \log n}$, the second term is $O(\delta \log g \log \log g) \subseteq O(\delta \log \frac{n \log \sigma}{\delta \log n} \log g) \subseteq O(\delta \log \frac{n \log \sigma}{\delta \log n} \log(\delta \log \frac{n \log \sigma}{\delta \log n}))$. The total expected construction time is $O(n \log n)$ [6]. This completes the proof of Theorem 1.3.

# 7 Conclusions and Future Work

We have obtained the best of two worlds [6, 7] in repetitive text indexing: an index of asymptotically optimal size, $O(\delta \log \frac{n \log \sigma}{\delta \log n})$, with nearly-optimal search time, $O(m+(occ+1)\log^\epsilon n)$, where $n$ is the text size, $\delta$ its repetitiveness measure, $m$ is the pattern length, *occ* is the number of times the pattern occurs in the text, and $\epsilon > 0$ is any constant. This closes a question open in those previous works. We also manage to search in optimal $O(m + occ)$ time within near-optimal space, $O(\delta \log \frac{n \log \sigma}{\delta \log n} \log^\epsilon(\delta \log \frac{n \log \sigma}{\delta \log n}))$. Finally, we show how to count the occurrences of the pattern in the text without enumerating them: we can do this in $O(m + \log^{2+\epsilon} n)$ time and optimal $O(\delta \log \frac{n \log \sigma}{\delta \log n})$ space, or in optimal $O(m)$ time and $O(\delta \log \frac{n \log \sigma}{\delta \log n} \log(\delta \log \frac{n \log \sigma}{\delta \log n}))$ space.

An important open question is whether this final gap can be closed: can we search or count in optimal time within δ-optimal space, $O(\delta \log \frac{n \log \sigma}{\delta \log n})$? With our best results, space or time must be multiplied by $O(\log^\epsilon n)$ or $O(\log n)$. Further improvements of the search time could also be possible in the packed setting, where the pattern can be read in $O(\lceil \frac{m \log \sigma}{\log n} \rceil)$ time.

Further research goes in the direction of providing more complex search functionality within δ-bounded space. For example, a recent article [23] shows how to efficiently find the maximal exact matches (MEMs) of a pattern in the text, within $O(\delta \log \frac{n}{\delta})$ space. The way towards a final goal like providing suffix tree functionality in δ-bounded space has several intermediate problems like this one, which is of relevance in bioinformatics applications. The very recent δ-SA [28], developed after the submission of this work, supports polylogarithmic-time suffix array and inverse suffix array queries in $O(\delta \log \frac{n \log \sigma}{\delta \log n})$ space, and thus constitutes a major step in that direction.

## Competing interests

All authors certify that they have no affiliations with or involvement in any organization or entity with any financial interest or non-financial interest in the subject, matter, or materials discussed in this manuscript.

# References

[1] Stephens, Z.D., Lee, S.Y., Faghri, F., Campbell, R.H., Zhai, C., Efron, M.J., Iyer, R., Schatz, M.C., Sinha, S., Robinson, G.E.: Big data: Astronomical or genomical? PLoS Biology **13**(7), 1002195 (2015). https://doi.org/10.1371/journal.pbio.1002195

[2] Navarro, G.: Indexing highly repetitive string collections, part II: Compressed indexes. ACM Computing Surveys **54**(2), 26–12632 (2021). https://doi.org/10.1145/3432999

[3] Navarro, G.: Indexing highly repetitive string collections, part I: Repetitiveness measures. ACM Computing Surveys **54**(2), 29–12931 (2021).

https://doi.org/10.1145/3434399

[4] Kreft, S., Navarro, G.: On compressing and indexing repetitive sequences. Theoretical Computer Science **483**, 115–133 (2013). https://doi.org/10.1016/j.tcs.2012.02.006

[5] Kempa, D., Prezza, N.: At the roots of dictionary compression: string attractors. In: 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, pp. 827–840 (2018). https://doi.org/10.1145/3188745.3188814

[6] Christiansen, A.R., Ettienne, M.B., Kociumaka, T., Navarro, G., Prezza, N.: Optimal-time dictionary-compressed indexes. ACM Transactions on Algorithms **17**(1), 8–1839 (2021). https://doi.org/10.1145/3426473

[7] Kociumaka, T., Navarro, G., Prezza, N.: Towards a definitive compressibility measure for repetitive sequences. IEEE Transactions on Information Theory (2022). https://doi.org/10.1109/TIT.2022.3224382

[8] Lempel, A., Ziv, J.: On the complexity of finite sequences. IEEE Transactions on Information Theory **22**(1), 75–81 (1976). https://doi.org/10.1109/TIT.1976.1055501

[9] Claude, F., Navarro, G.: Improved grammar-based compressed indexes. In: 19th International Symposium on String Processing and Information Retrieval, SPIRE 2012. LNCS, vol. 7608, pp. 180–192 (2012). https://doi.org/10.1007/978-3-642-34109-0_19

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

[11] Mehlhorn, K., Sundar, R., Uhrig, C.: Maintaining dynamic sequences under equality tests in polylogarithmic time. Algorithmica **17**(2), 183–198 (1997). https://doi.org/10.1007/BF02522825

[12] Jeż, A.: A really simple approximation of smallest grammar. Theoretical Computer Science **616**, 141–150 (2016). https://doi.org/10.1016/j.tcs.2015.12.032

[13] Kociumaka, T., Navarro, G., Olivares, F.: Near-optimal search time in δ-optimal space. In: 15th Latin American Symposium on Theoretical Informatics, LATIN 2022. LNCS, vol. 13568, pp. 88–103 (2022). https://doi.org/10.1007/978-3-031-20624-5_6

[14] Batu, T., Sahinalp, S.C.: Locally consistent parsing and applications to approximate string comparisons. In: 9th International Conference on

Developments in Language Theory, DLT 2005. LNCS, vol. 3572, pp. 22–35 (2005). https://doi.org/10.1007/11505877_3

[15] Cole, R., Vishkin, U.: Deterministic coin tossing and accelerating cascades: Micro and macro techniques for designing parallel algorithms. In: 18th Annual ACM Symposium on Theory of Computing, STOC 1986, pp. 206–219 (1986). https://doi.org/10.1145/12130.12151

[16] Raskhodnikova, S., Ron, D., Rubinfeld, R., Smith, A.D.: Sublinear algorithms for approximating string compressibility. Algorithmica **65**(3), 685–709 (2013). https://doi.org/10.1007/s00453-012-9618-6

[17] Kociumaka, T., Radoszewski, J., Rytter, W., Waleń, T.: Internal pattern matching queries in text and applications (2023) arXiv:1311.6235v5

[18] Birenzwige, O., Golan, S., Porat, E.: Locally consistent parsing for text indexing in small space. In: 31st Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, pp. 607–626 (2020). https://doi.org/10.1137/1.9781611975994.37

[19] Kempa, D., Kociumaka, T.: Dynamic suffix array with polylogarithmic queries and updates. In: 54th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2022, pp. 1657–1670 (2022). https://doi.org/10.1145/3519935.3520061

[20] Sahinalp, S.C., Vishkin, U.: On a parallel-algorithms method for string matching problems (overview). In: 2nd Italian Conference on Algorithms and Complexity, CIAC 1994. LNCS, vol. 778, pp. 22–32 (1994). https://doi.org/10.1007/3-540-57811-0_3

[21] Chan, T.M., Larsen, K.G., Pătraşcu, M.: Orthogonal range searching on the RAM, revisited. SoCG '11, pp. 1–10. Association for Computing Machinery, New York, NY, USA (2011). https://doi.org/10.1145/1998196.1998198

[22] Karp, R.M., Rabin, M.O.: Efficient randomized pattern-matching algorithms. IBM Journal of Research and Development **31**(2), 249–260 (1987). https://doi.org/10.1147/rd.312.0249

[23] Navarro, G.: Computing MEMs on repetitive text collections. In: 34th Annual Symposium on Combinatorial Pattern Matching, CPM 202. LIPIcs, vol. 259, pp. 24–12417 (2023). https://doi.org/10.4230/LIPIcs.CPM.2023.24

[24] Fredman, M.L., Komlós, J., Szemerédi, E.: Storing a sparse table with $O(1)$ worst case access time. J. ACM **31**(3), 538–544 (1984). https://doi.org/10.1145/828.1884

[25] Alstrup, S., Brodal, G.S., Rauhe, T.: New data structures for orthogonal range searching. Proceedings 41st Annual Symposium on Foundations of Computer Science, 198–207 (2000). https://doi.org/10.1109/SFCS.2000.892088

[26] Fine, N.J., Wilf, H.S.: Uniqueness theorems for periodic functions. Proceedings of the American Mathematical Society **16**(1), 109–114 (1965). https://doi.org/10.1090/s0002-9939-1965-0174934-9

[27] Belazzougui, D., Navarro, G.: Alphabet-independent compressed text indexing. ACM Transactions on Algorithms **10**(4), 23–12319 (2014). https://doi.org/10.1145/2635816

[28] Kempa, D., Kociumaka, T.: Collapsing the hierarchy of compressed data structures: Suffix arrays in optimal compressed space. In: Proc. 64th Annual IEEE Symposium on Foundations of Computer Science (FOCS) (2023). https://doi.org/10.48550/arXiv.2308.03635