Near-Optimal Search Time in δ-Optimal Space[⋆]

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

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

Abstract. Two recent lower bounds on the compressiblity of repetitive sequences, $\delta \leq \gamma$, have received much attention. It has been shown that a string $S[1..n]$ can be represented within the optimal $O(\delta \log \frac{n}{\delta})$ space, and further, that within that space one can find all the *occ* occurrences in S of any pattern of length m in time $O(m \log n + occ \log^{\epsilon} n)$ for any constant $\epsilon > 0$. Instead, the near-optimal search time $O(m+(occ+1)\log^{\epsilon} n)$ was 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 obtained within the δ-optimal space was open. In this paper, we prove that both techniques can indeed be combined in order to obtain the best of both worlds, $O(m + (occ+1)\log^{\epsilon} n)$ search time within $O(\delta \log \frac{n}{\delta})$ space.

1 Introduction

The amount of data we are expected to handle has been growing steadily in the last decades [20]. The fact that much of the fastest-growing data is composed of highly repetitive sequences has raised the interest in text indexes whose size can be bounded by some measure of repetitiveness [17], and in the study of those repetitiveness measures [16]. Since statistical compression does not capture repetitiveness well [13], 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 γ of the smallest string attractor [9] and the substring complexity δ [3,10]. It holds that $\delta \leq \gamma$ for every string [3] (with $\delta = o(\gamma)$ in some string families [11]), and that γ asymptotically lower-bounds a number of other measures sensitive to repetitiveness [9] (e.g., the size of the smallest Lempel–Ziv parse [14]). On the other hand, any string $S[1..n]$ can be represented within $O(\delta \log \frac{n}{\delta})$ space, and this bound is tight for every n and δ [18,10,11].

A more ambitious goal than merely representing S in compressed space is to *index* it within that space so that, given any pattern $P[1..m]$, 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 tight $O(\delta \log \frac{n}{\delta})$ space, so as to search for P in time $O(m \log n + occ \log^{\epsilon} n)$ time [10,11]. If one allows the higher $O(\gamma \log \frac{n}{\gamma})$ space, the search time can be reduced to $O(m + (occ+1)\log^{\epsilon} n)$ [3],

[⋆] Funded in part by Basal Funds FB0001, Fondecyt Grant 1-200038 and a Conicyt Doctoral Scholarship, ANID, Chile.

which is optimal in terms of the pattern length and near-optimal in the time per reported occurrence. Within (significantly) more space, $O(\gamma \log \frac{n}{\gamma} \log n)$, one obtains truly optimal search time, $O(m + occ)$.

The challenge of obtaining the near-optimal search time $O(m+(occ+1)\log^{\epsilon} n)$ within tight space $O(\delta \log \frac{n}{\delta})$ was posed [3,10,11], and this is what we settle on the affirmative in this paper. Both previous results build a convenient context-free grammar on S and then adapt a classical grammar-based index on it [4,5]. The index based on attractors [3] constructs a grammar from a locally consistent parsing [15] of S that forms blocks in S ending at every minimum of a randomized mapping on the alphabet, collapsing every block into a nonterminal and iterating. The smaller grammar based on substring complexity [11] uses another locally consistent parsing called recompression [7], 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 δ-bounded space is to pause the pairing on symbols that become too long for the iteration where they were formed [10,11]. We show that the pausing idea can be applied to the first kind of locally consistent grammar as well and that, although it leads to possibly larger grammars, it still yields the desired time and space complexities. The next theorem summarizes our result.

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

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 $\Sigma = \{1, \ldots, \sigma\}$. We denote as $\Sigma(S)$ the subset of Σ consisting of symbols that occur in S. The length of S is denoted $|S| = n$. 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' is the empty string ε or $S' = S[i..j] = S[i] \cdots S[j]$ for some $1 \leq i \leq j \leq n$. We also use "(" and ")" to denote non-inclusive intervals: $S(i..j) = S[i + 1..j - 1]$, $S(i..j] = S[i + 1..j]$, and $S[i..j) = S[i..j - 1]$. With the term *fragment*, we refer to a particular occurrence $S[i..j]$ of a substring in S (not just the substring content). We use S^{rev} to denote the reverse of S, that is, $S^{rev} = S[n] \cdot S[n - 1] \cdots S[1]$.

We use the RAM model of computation with word size $w = \Theta(\log n)$ bits. By default, we measure the space in words, which means that $O(x)$ space comprises of $O(x \log n)$ bits.

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 nonterminals 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 $A \rightarrow A_1^m$, where A, A_1 are nonterminals and $m \in \mathbb{Z}_{\geq 2}$, which means that A is a rule composed by concatenating m 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)* [1] is a parsing where, if two fragments $S[i..j] = S[i'..j']$ appear inside equal long enough contexts $S[i-\alpha..j+\beta] = S[i'-\alpha..j'+\beta]$, then the same blocks are formed inside $S[i..j]$ and $S[i'..j']$. The meaning of "long enough" depends on the type of LCP [1,6,3].

3 A New δ-bounded RLSLP

The measure δ was originally introduced in a stringology context [18], but it was formally defined later [3] as a way to construct a grammar of size $O(\gamma \log \frac{n}{\gamma})$ without knowing γ. For a given string $S[1..n]$, 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, δ is defined as

$$\delta \; = \; \max\left\{\tfrac{d_k(S)}{k} : k \in [1..n]\right\}.$$

An RLSLP of size $O(\delta \log \frac{n}{\delta})$ was built [11] on top of the recompression method [7]. In this section, we show that the same can be achieved on top of the block-based LCP [15]. Unlike the previous construction, ours produces an RLSLP with $O(\delta \log \frac{n}{\delta})$ rules in $O(n)$ deterministic time, though we still need randomization in order to ensure that the total grammar size is also $O(\delta \log \frac{n}{\delta})$.

We adapt the preceding construction [11], which uses the so-called *restricted recompression* [12]. This technique pauses the processing for symbols whose expansion is too long for the current stage. A similar idea was used [2,8] for adapting another LCP, called *signature parsing* [19]. We apply restriction (the pausing technique) to the LCP of [15] that forms blocks ending at local minima of a randomized bijective function, which is interpreted as an alphabet permutation. This LCP will be used later to obtain near-optimal search time, extending previous work [3]. 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 Σ, 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 *non-terminals*, which are naturally associated with *productions* $(A_1, \ldots, A_j) \to A_1 \cdots A_j$ for $(A_1, \ldots, A_j) \in \mathcal{A}^{\geq 2}$ and $(A_1, m) \to A_1^m$ for $(A_1, m) \in \mathcal{A} \times \mathbb{Z}_{\geq 2}$. Setting any $A \in \mathcal{A}$ as the starting symbol yields an RLSLP. The string generated by this RLSLP is $\texttt{exp}(A)$, where $\texttt{exp} : \mathcal{A} \to \Sigma^+$ is the *expansion* function defined recursively:

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

Then, for every string $(S_k)_{k \geq 0}$ generated using restricted block compression, if the expansion function is extended homomorphically to $\texttt{exp} : \mathcal{A}^* \to \Sigma^*$, with $\texttt{exp}(A_1 \cdots A_m) = \texttt{exp}(A_1) \cdots \texttt{exp}(A_m)$ for $A_1 \cdots A_m \in \mathcal{A}^*$, then it must hold that $\texttt{exp}(S_k) = S$ for every $k \in \mathbb{Z}_{\geq 0}$. Starting from $S_0 = S$, the strings $(S_k)_{k \geq 1}$ 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 $m \geq 2$ into individual symbols in $\mathcal{A}$. In Definition 3.1, 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 3.1 (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 $i \in [1..|T|)$, 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 each block $T[i..i+m)$ of $m \geq 2$ equal symbols A, replace $T[i..i+m) = A^m$ with the symbol $(A, m) \in \mathcal{A}$.*

In Definition 3.3, the blocks boundaries are determined by local minima of a permutation on $\mathcal{A}$, and the blocks are collapsed to symbols in $\mathcal{A}^{\geq 2}$.

Definition 3.2 (Local minima). *Given $T \in \mathcal{A}^+$ and a bijective function $\pi : \Sigma(T) \to [1..|\Sigma(T)|]$, we say that $j \in (1..|T|)$ is a* local minimum *if*

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

Definition 3.3 (Restricted block parsing). *Given $T \in \mathcal{A}^+$, a bijective function $\pi : \Sigma(T) \to [1..|\Sigma(T)|]$, 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 $i \in [1..|T|)$, 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 mimimum with respect to π.*
2) *For each block $T[i..i+m)$ of length $m \geq 2$, replace $T[i..i+m)$ with a symbol $(T[i], \ldots, T[i+m-1]) \in \mathcal{A}$.*

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 have become too long for that level.

Definition 3.4 (Restricted block compression). *Given a string $S \in \Sigma^+$, the strings S_k for $k \in \mathbb{Z}_{\geq 0}$ are constructed as follows, where $\ell_k := \left(\frac{4}{3}\right)^{\lceil k/2 \rceil - 1}$, $\mathcal{A}_k := \{A \in \mathcal{A} : |\mathtt{exp}(A)| \leq \ell_k\}$, and $\pi_k : \Sigma(S_{k-1}) \to [1..|\Sigma(S_{k-1})|]$ is a bijection satisfying $\pi_k(A) < \pi_k(B)$ for every $A \in \Sigma(S_{k-1}) \setminus \mathcal{A}_k$ 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 $S_k = bc_{\pi_k, \mathcal{A}_k}(S_{k-1})$.*

Note that $\mathtt{exp}(S_k) = S$ holds for all $k \in \mathbb{Z}_{\geq 0}$.

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 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 $\mathcal{S}_k := \{S_k[j] : j \in [1..|S_k|]\}$ and $\mathcal{S} := \bigcup_{k=0}^{\infty} \mathcal{S}_k$.

We first prove an upper bound on $|S_k|$ which, in particular, implies that $|S_k| = 1$ holds after $O(\log n)$ iterations.

Lemma 3.5. *For every $k \in \mathbb{Z}_{\geq 0}$, we have $|S_k| < 1 + \frac{4n}{\ell_{k+1}}$.*

Proof. We proceed by induction on k. For $k = 0$, we have $|S_0| = n < 1 + 4n = 1 + \frac{4n}{\ell_1}$. If k is odd, we note that $|S_k| \leq |S_{k-1}| < 1 + \frac{4n}{\ell_k} = 1 + \frac{4n}{\ell_{k+1}}$. If k is even, let us define

$$J = \{j \in [1..|S_{k-1}|] : S_{k-1}[j] \notin \mathcal{A}_k\}.$$

Since $A \notin \mathcal{A}_k$ implies $|\mathtt{exp}(A)| > \ell_k$, we have $|J| < \frac{n}{\ell_k}$. Then, since no two consecutive symbols can be local minima, we have

$$\begin{aligned} |S_k| \leq 2|J| + 1 + \tfrac{|S_{k-1}| - (2|J|+1)}{2} = \tfrac{1 + |S_{k-1}|}{2} + |J| < 1 + \tfrac{2n}{\ell_k} + \tfrac{n}{\ell_k} &= 1 + \tfrac{3n}{\ell_k} \\ &= 1 + \tfrac{4n}{\ell_{k+1}}. \quad \square \end{aligned}$$

Our next goal is to prove that restricted block compression is a locally consistent parsing. For 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 $j \in [1..|S_k|]$, we define the level-k* phrases *of S induced by S_k as the fragments*

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

We also define the set B_k of phrase boundaries *induced by S_k:*

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

Lemma 3.7. *Consider integers $k, m, \alpha \geq 0$ with $\alpha \geq 8\ell_k$, as well as positions $i, i' \in [m + 2\alpha..n - \alpha]$ such that $S(i - m - 2\alpha..i + \alpha] = S(i' - m - 2\alpha..i' + \alpha]$.*

1) If $i \in B_k$, then $i' \in B_k$.
2) If $S(i - m..i]$ is a level-k phrase, then $S(i' - m..i']$ is a level-k phrase corresponding to the same symbol in S_k.

Proof. We proceed by induction on k, with a weaker assumption $\alpha \geq 7\ell_k$ for odd k. In the base case of $k = 0$, the claim is trivial because $B_k = [1..n)$ and $S_k = S$. Next, we prove that the claim holds for integers $k > 0$ and $\alpha > \ell_k$ assuming that it holds for all $k - 1$ and $\alpha - \lfloor \ell_k \rfloor$. This is sufficient for the inductive step: If $\alpha \geq 8\ell_k$ for even $k > 0$, then $\alpha - \lfloor \ell_k \rfloor \geq 7\ell_k = 7\ell_{k-1}$. Similarly, if $\alpha \geq 7\ell_k$ for odd k, then $\alpha - \lfloor \ell_k \rfloor \geq 6\ell_k = 8\ell_{k-1}$.

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 $i' \notin B_k$ for some $i, i' \in [2\alpha..n - \alpha]$. By the inductive assumption (applied to positions i, i'), $i \in B_k \subseteq B_{k-1}$ implies $i' \in B_{k-1}$. Let us set $j, j' \in [1..|S_{k-1}|)$ so that $i = |\mathtt{exp}(S_{k-1}[1..j])|$ and $i' = |\mathtt{exp}(S_{k-1}[1..j'])|$. By the assumptions on i, i', 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 3.1 and 3.3, the latter implies $S_{k-1}[j'], S_{k-1}[j'+1] \in \mathcal{A}_k$. Consequently, the phrases $S(i'-\ell..i'] = \mathtt{exp}(S_{k-1}[j'])$ and $S(i'..i'+r] = \mathtt{exp}(S_{k-1}[j'+1])$ around position i' are of length at most $\lfloor \ell_k \rfloor$. Since $i' - \lfloor \ell_k \rfloor \leq i' - \ell \leq i' + r \leq i' + \lfloor \ell_k \rfloor$, the inductive assumption applied to positions i', i and $i' + r, i + r$ implies that $S(i - \ell..i]$ and $S(i..i + r]$ are 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 places a boundary between $S_{k-1}[j], S_{k-1}[j + 1] \in \mathcal{A}_k$, we conclude from Definition 3.3 that j must be a local minimum with respect to π_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 π_k imposed in Definition 3.4 implies $S_{k-1}[j - 1] \in \mathcal{A}_k$. Consequently, the phrase $S(i - \ell'..i - \ell] = \mathtt{exp}(S_{k-1}[j - 1])$ is of length at most $\lfloor \ell_k \rfloor$. Since $i' - 2\lfloor \ell_k \rfloor \leq i' - \ell' \leq i' - \ell \leq i'$, the inductive assumption, applied to positions $i - \ell, i' - \ell$ implies that $S(i' - \ell'..i' - \ell]$ is 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 π_k and, by Definition 3.3, contradicts $i' \notin B_k$.

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, $S(i' - m..i']$ 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 first item, the parsing of S_{k-1} places block boundaries before $S_{k-1}[j' - m']$ and after $S_{k-1}[j']$, but nowhere in between. Consequently, $S_{k-1}(j - m'..j]$ and

$S_{k-1}(j'-m'..j']$ are matching blocks, which means that they are collapsed into matching symbols of S_k, Thus, the level-k phrases $S(i-m..i]$ and $S(i'-m..i']$ are represented by matching symbols in S_k. □

Our next goal is to prove that $|\mathcal{S}| = O(\delta \log \frac{n}{\delta})$ (Corollary 3.12). As a first step, we show that $|\mathcal{A}_{k+1} \cap \mathcal{S}_k| = O(\delta)$ (Lemma 3.9). The idea for this proof is to consider the leftmost occurrence of all symbols of S_k and then bound the set of those occurrences in relation to δ (Claims 3.10 and 3.11). At a high level, we build on the arguments of [11], where the same bound was proved in expectation, but we obtain worst-case results with our parsing. We start by generalizing Lemma 3.5.

Lemma 3.8. *For every $k \in \mathbb{Z}_{\geq 0}$ and every interval $I \subseteq [1..n]$, we have*

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

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

$$J = \{j \in [1..|S_{k-1}|] : S_{k-1}[j] \notin \mathcal{A}_k\},$$
$$J_I = \{j \in J : |\mathtt{exp}(S_{k-1}[1..j))| \in I\} \subseteq B_{k-1} \cap I.$$

Since $A \notin \mathcal{A}_k$ implies $|\mathtt{exp}(A)| > \ell_k$, we have $|J_I| < \frac{|I|}{\ell_k}$. Then, since no two consecutive symbols can be local minima, we have

$$|B_k \cap I| \leq 2|J_I| + 1 + \tfrac{|B_{k-1}\cap I| - (2|J_I|+1)}{2} = \tfrac{1+|B_{k-1}\cap I|}{2} + |J_I|$$
$$< 1 + \tfrac{2|I|}{\ell_k} + \tfrac{|I|}{\ell_k} = 1 + \tfrac{3|I|}{\ell_k} = 1 + \tfrac{4|I|}{\ell_{k+1}}. \quad □$$

The following result is used to bound both the number of symbols $|\mathcal{S}|$ (where we only care about $|\mathcal{S}_k \cap \mathcal{A}_{k+1}|$, i.e., the number of substrings with $m = 1$ active symbol) and the size of the RLSLP resulting from restricted block compression.

Lemma 3.9. *If the string S has measure δ, then, for all integers $k \geq 0$ and $m \geq 1$, the string S_k contains $O(m\delta)$ distinct length-m substrings in $\mathcal{A}_{k+1}^*$.*

Proof. Denote $\alpha := \lceil 8\ell_k \rceil$ and $\ell := 3\alpha + \lfloor m\ell_{k+1} \rfloor$, and let L be the set of positions in S covered by the leftmost occurrences of substrings of S of length at most ℓ, as well as the trailing ℓ positions in S. We first prove two auxiliary claims.

Claim 3.10. *The string S_k contains at most $|L \cap B_k|$ distinct length-m substrings in $\mathcal{A}_{k+1}^*$.*

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

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

A level-k phrase ends at position q, so we also have $q \in B_k$. Since the position q is uniquely determined by the substring T, this yields an upper bound of $|L \cap B_k|$ on the number of choices for T. □

Claim 3.11. *The set L forms $O(\delta)$ intervals of total length $O(\delta\ell)$.*

Proof. Each position in $L \cap (0..n-\ell]$ is covered by the leftmost occurrence of a substring of length exactly ℓ, and thus L forms at most $\lfloor \frac{1}{\ell}|L| \rfloor$ intervals of length at least ℓ each. Hence, it suffices to prove that the total length satisfies $|L| = O(\delta\ell)$. For this, note that, for each position $j \in L \cap [\ell..n-\ell]$, the fragment $S(j-\ell..j+\ell]$ is the leftmost occurrence of a length-2ℓ substring of S; this because any length-ℓ fragment covering position j is contained within $S(j-\ell..j+\ell]$. Consequently, $|L| \le d_{2\ell}(S) + 2\ell = O(\delta\ell)$ holds as claimed. □

By Claim 3.10, it remains to prove that $|L \cap B_k| = O(\delta m)$. Let $\mathcal{I}$ be the family of intervals covering L. For each $I \in \mathcal{I}$, Lemma 3.8 implies $|B_k \cap I| \le 1 + \frac{4|I|}{\ell_{k+1}}$. By the bounds on $\mathcal{I}$ following from Claim 3.11, this yields the announced result:

$$|B_k \cap L| \le |\mathcal{I}| + \tfrac{4}{\ell_{k+1}} \sum_{I \in \mathcal{I}} |I| = O(\delta + \tfrac{\delta\ell}{\ell_{k+1}}) = O(\delta m). \qquad \square$$

The proof of our main bound $|\mathcal{S}| = O(\delta \log \frac{n}{\delta})$ combines Lemmas 3.5 and 3.9.

Corollary 3.12. *For every string S of length n and measure δ, we have $|\mathcal{S}| = O(\delta \log \frac{n}{\delta})$.*

Proof. Note that $|\mathcal{S}| \le 1 + \sum_{k=0}^{\infty} |\mathcal{S}_k \setminus \mathcal{S}_{k+1}|$. We combine two upper bounds on $|\mathcal{S}_k \setminus \mathcal{S}_{k+1}|$, following from Lemmas 3.5 and 3.9, respectively.

First, we observe that Definition 3.4 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 length-1 substring of S_{k+1}, and thus $|\mathcal{S}_k \setminus \mathcal{S}_{k+1}| \le |\mathcal{S}_k \cap \mathcal{A}_{k+1}| = O(\delta)$ holds due to Lemma 3.9. Secondly, we note that $|\mathcal{S}_k \setminus \mathcal{S}_{k+1}| = 0$ if $|S_k| = 1$ and $|\mathcal{S}_k \setminus \mathcal{S}_{k+1}| \le |S_k| \le 2(|S_k|-1)$ if $|S_k| \ge 2$. Hence, Lemma 3.5 yields

$$|\mathcal{S}_k \setminus \mathcal{S}_{k+1}| \le 2(|S_k|-1) \le \tfrac{8n}{\ell_{k+1}} = O((\tfrac{3}{4})^{k/2} n).$$

We apply the first or the second upper bound on $|\mathcal{S}_k \setminus \mathcal{S}_{k+1}|$ depending on whether $k < \lambda := 2\lfloor \log_{4/3} \frac{n}{\delta} \rfloor$. This yields

$$\sum_{k=0}^{\infty} |\mathcal{S}_k \setminus \mathcal{S}_{k+1}| = \sum_{k=0}^{\lambda-1} O(\delta) + \sum_{k=\lambda}^{\infty} O((\tfrac{3}{4})^{k/2} n)$$
$$= 2\lfloor \log_{4/3} \tfrac{n}{\delta} \rfloor \cdot O(\delta) + \sum_{i=0}^{\infty} O((\tfrac{3}{4})^{i/2}\delta) = O(\delta \log \tfrac{n}{\delta}).$$

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

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

Theorem 3.13. *Consider the restricted block compression of a string $S[1..n]$ with measure δ, where the functions $(\pi_k)_{k \geq 0}$ in Definition 3.4 are chosen uniformly at random. Then, the expected size of the resulting RLSLP is $O(\delta \log \frac{n}{\delta})$.*

Proof. Although Corollary 3.12 guarantees that $|\mathcal{S}| = O(\delta \log \frac{n}{\delta})$, the remaining problem is that the size of the resulting grammar (i.e., 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 $k > 0$. In order to estimate their contribution to grammar size, we shall fix $\pi_0, \ldots, \pi_k$ and compute the expectation with respect to the random choice of π_{k+1}. In this setting, we prove the following claim:

Claim 3.14. *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 $t \in [1..m]$ 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 π_{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 3.9, the string S_k has $O(\delta m)$ distinct substrings $T \in \mathcal{A}_{k+1}^m$. By Claim 3.14, 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, $\sum_{m=2}^{\infty} O(m \cdot \delta m \cdot 2^{-m}) = O(\delta)$.

At the same time, $\mathcal{S}_{k+1} \setminus \mathcal{S}_k = \emptyset$ if $|S_k| = 1$ and, if $|S_k| \geq 2$, the contribution of symbols in $\mathcal{S}_{k+1} \setminus \mathcal{S}_k$ to the RLSLP size is most $|S_k| \leq 2(|S_k| - 1) \leq \frac{8n}{\ell_{k+1}} = O((\frac{3}{4})^{k/2} n)$, where the bound on $|S_k|$ follows from Lemma 3.5. This sums up to $O(\delta)$ across all odd levels $k > \lambda := 2\lfloor \log_{4/3} \frac{n}{\delta} \rfloor$. Overall, we conclude that the total expected RLSLP size is $O(\delta \log \frac{n}{\delta} + (\lambda + 1)\delta) = O(\delta \log \frac{n}{\delta})$. □

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

Corollary 3.15. *Given $S[1..n]$ with measure δ, we can build an RLSLP of size $O(\delta \log \frac{n}{\delta})$ in $O(n)$ expected time.*

Proof. We apply Definition 3.4 on top of the given string S, with functions π_k choices uniformly at random. It is an easy exercise to carry out this construction in $O(\sum_{k \geq 0} |S_k|) = O(n)$ worst-case time.

The expected size of the resulting RLSLP is $c \cdot \delta \log \frac{n}{\delta}$ for some constant c; we can repeat the construction (with fresh randomness) until it yields an RLSLP of size at most $2c \cdot \delta \log \frac{n}{\delta}$. 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 $O(\delta \log \frac{n}{\delta})$. □

Remark 3.16 (Grammar height). In the algorithm of Corollary 3.15, 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 $\texttt{exp}(A_\lambda) = S$. Lemma 3.5 yields $|S_\lambda| = O(1 + (\frac{3}{4})^{\lambda/2} n) = O(\delta)$, so the resulting RLSLP is still of size $O(\delta \log \frac{n}{\delta})$; 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 [3] achieves this by showing that, thanks to the locally consistent parsing, only a set $M(P)$ of $O(\log |P|)$ pattern positions need 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 $S[i..j]$ of S, we define*

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

and

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

where $\alpha_k = \lceil 8\ell_k \rceil$ and $\{\min \emptyset\} = \emptyset$.

Intuitively, the set $B_k(i,j)$ lists (the relative locations of) all level-k phrase boundaries inside $S[i..j]$. 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 $S[i..j]$ (in the light of Lemma 3.7, it may depend on the context of $S[i..j]$ which of these phrase boundaries are preserved in level $k+1$) as well as the leftmost phrase boundary within the remaining internal part of $S[i..j]$.

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 $S[i'..j'] = S[i..j]$, then $M(i',j') = M(i,j)$.*

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

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

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 $[2\alpha_{k+1}..j-i-\alpha_{k+1})$ 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 $q \in B_k(i,j) \cap [0..2\alpha_{k+1})$ corresponds to $q + i \in B_k \cap [i..i+2\alpha_{k+1})$ and each element $q \in B_k(i,j) \cap [j-i-\alpha_{k+1}..j-i)$ corresponds to $q+i \in B_k \cap [j-\alpha_{k+1}..j)$. By Lemma 3.8, we conclude that $|M_k(i,j)| \leq 1 + (1 + \frac{8\alpha_{k+1}}{\ell_{k+1}}) + (1 + \frac{4\alpha_{k+1}}{\ell_{k+1}}) = O(1)$. Moreover, if $\ell_k > 4(j-i)$, then Lemma 3.8 further yields $|B_k(i,j)| = |B_k \cap [i..j)| \leq 1$. 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 indices 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 adapt the results on attractors [3, Sec. 6] to our modified parsing, so as to obtain our main result.

Definition 5.1 ([3]). *The grammar tree of a RLCFG 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 its total number of nodes is the grammar size plus one. We identify each nonterminal A with the only internal grammar tree node labeled A. We also sometimes identify terminal symbols a with grammar tree leaves.

The search algorithm classifies the occurrences of a pattern $P[1..m]$ in S into "primary" and "secondary", according to the partition of S induced by the grammar tree leaves.

Definition 5.2 ([3]). *The leaves of the grammar tree induce a partition of S into phrases. An occurrence of $P[1..m]$ at $S[t..t+m)$ is primary if the lowest grammar tree node deriving a range of S that contains $S[t..t+m)$ is internal (or, equivalently, the occurrence crosses the boundary between two phrases); otherwise it is secondary.*

The general idea of the search is to find the primary occurrences by looking for prefix-suffix partitions of P and then find the secondary occurrences from the primary ones [5].

5.1 Finding the primary occurrences

Let nonterminal A be the lowest (internal) grammar tree node that covers a primary occurrence $S[t..t+m)$ of $P[1..m]$. Then, if $A \to A_1 \cdots A_s$, there exists some $i \in [1..s)$ and $q \in [1..m)$ such that (1) a suffix of $\texttt{exp}(A_i)$ matches $P[1..q]$, and (2) a prefix of $\texttt{exp}(A_{i+1}) \cdots \texttt{exp}(A_s)$ matches $P(q..m]$. The idea is to index all the pairs $(\texttt{exp}(A_i)^{rev}, \texttt{exp}(A_{i+1}) \cdots \texttt{exp}(A_s))$ and find those where the first and second component are prefixed by $(P[1..q])^{rev}$ and $P(q..m]$, respectively. Note that there is exactly one such pair per border between two consecutive phrases (or leaves in the grammar tree).

Definition 5.3 ([3]). *Let v be the lowest (internal) grammar tree node that covers a primary occurrence $S[t..t+m)$ of P. Let v_i be the leftmost child of v that overlaps $S[t..t+m)$. We say that node v is the parent of the primary occurrence $S[t..t+m)$ of P and node v_i is its locus.*

The index [3] builds a two-dimensional grid data structure. It lexicographically sorts all the components $\texttt{exp}(A_i)^{rev}$ to build the x-coordinates, and all the components $\texttt{exp}(A_{i+1}) \cdots \texttt{exp}(A_s)$ to build the y-coordinates; then, it fills the grid with points $(\texttt{exp}(A_i)^{rev}, \texttt{exp}(A_{i+1}) \cdots \texttt{exp}(A_s))$, each associated with the locus A_i. The size of this data structure is of the order of the number of points, which is bounded by the grammar size, $g = O(\delta \log \frac{n}{\delta})$ in our case. The structure can find all the p points within any orthogonal range in time $O((p+1)\log^{\epsilon} g)$, where $\epsilon > 0$ is any constant fixed at construction time.

Given a partition $P = P[1..q] \cdot P(q..m]$ to test, they search for $P[1..q]^{rev}$ in a data structure that returns the corresponding range in x, search for $P(q..m]$ in a similar data structure that returns the corresponding range in y, and then perform the corresponding range search on the geometric data structure.

They show [3, Sec. 6.3] that the x- and y-ranges of any τ cuts of P can be computed in time $O(m + \tau \log^2 m)$, within $O(g)$ space. All they need from the RLCFG to obtain this result is that (1) one can extract any length-ℓ prefix or suffix of any $\texttt{exp}(A)$ in time $O(\ell)$, which is proved for an arbitrary RLCFG; and (2) one can compute a Karp–Rabin fingerprint of any substring of S in time $O(\log^2 \ell)$, which is shown to be possible for any locally contracting grammar, which follows from our Lemma 3.8.

In total, if we have identified τ cuts of P that suffice to find all of its occurrences in S, then we can find all the $occ_p \leq occ$ primary occurrences of P in time $O(m + \tau(\log^{\epsilon} g + \log^2 m) + occ_p \log^{\epsilon} g)$.

5.2 Parsing the pattern

The next step is to set a bound for τ with our parsing and show how to find the corresponding cuts.

Lemma 5.4. *Using our grammar of Section 3, there are only $\tau = O(\log m)$ cuts $P = P[1..q] \cdot P(q..m]$ yielding primary occurrences of $P[1..m]$. These positions belong to $M(P)+1$ (see Definition 4.3).*

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

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

(2) $A \rightarrow A_1^s$ is a run-length nonterminal, and a suffix of $\texttt{exp}(A_1)$ matches $P[1..q]$, for some $q \in [1..m)$. 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, $|M(P)| = O(\log m)$. □

The parsing is done in $O(m)$ time almost exactly as in previous work [3, Sec. 6.1], with the difference that we have to care about paused symbols. Essentially, we store the permutations π_k drawn when indexing S and use them to parse P in the same way, level by level. We then work for $O(\log m)$ levels on exponentially decreasing sequences, in linear time per level, which adds up to $O(m)$. There are a few differences with respect to previous work, however [3]:

1) In the parsing of [3], the symbols are disjoint across levels, so the space to store the permutations π_k is proportional to the grammar size. In our case, instead, paused symbols exist along several consecutive levels and participate in several permutations. However, by Lemma 3.9, we have $|\mathcal{S}_k \cap \mathcal{A}_{k+1}| = O(\delta)$ active symbols in S_k. We store store the values of π_{k+1} only for these symbols and observe that the values π_{k+1} for the remaining symbols do not affect the placement of block boundaries in Definition 3.3: If $S_k[j], S_k[j+1] \in \mathcal{A}_{k+1}$, then, due condition imposed on π_{k+1} in Definition 3.4, 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 $A \in \Sigma(S_k) \setminus \mathcal{A}_{k+1}$ and obtain the same parsing of S. By storing the values of π_k only for the active symbols, we use $O(\delta \log \frac{n}{\delta})$ total space.
2) They use that the number of symbols in the parsing of P halve from a level to the next in order to bound the number of levels in the parse and the total amount of work. While this is not the case in our parsing with paused symbols, it still holds by Lemmas 3.5 and 3.8 that 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.

5.3 Secondary occurrences and short patterns

The occ_s secondary occurrences can be obtained in $O(occ_s)$ time given the primary ones, with a technique that works for any arbitrary RLCFG and within $O(g)$ space [3, Sec. 6.4]. Plugged with the preceding results, the total space of our index is $O(\delta \log \frac{n}{\delta})$ and its search time is $O(m + \tau(\log^\epsilon g + \log^2 m) + occ \log^\epsilon g) = O(m + \log^\epsilon g \log m + occ \log^\epsilon g)$. This bound exceeds $O(m + (occ + 1) \log^\epsilon g)$ only when $m = O(\log^\epsilon g \log\log g)$. In that case, however, the middle term is $O(\log^\epsilon g \log\log g)$, which becomes $O(\log^\epsilon g)$ again if we infinitesimally adjust ϵ.

The final touch is to reduce the $O(m + \log^\epsilon g + occ \log^\epsilon g)$ 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 $\log^\epsilon g \in O(\log^\epsilon(\delta \log \frac{n}{\delta})) \subseteq O(\log^\epsilon \delta + \log\log \frac{n}{\delta})$, so it suffices to detect in $O(m)$ time the patterns of length $m \leq \ell = \log\log \frac{n}{\delta}$ that do not occur in S. By definition of δ, there are at most $\delta\ell$ strings of length ℓ in S, so we can store them all in a trie using total space $O(\delta\ell^2) \subseteq O(\delta \log \frac{n}{\delta})$. By implementing the trie children with perfect hashing, we can verify in $O(m)$ time whether a pattern of length $m \leq \ell$ occurs in S. We then obtain Theorem 1.1.

6 Conclusions and Future Work

We have obtained the best of two worlds [3,10] in repetitive text indexing: an index of asymptotically optimal size, $O(\delta \log \frac{n}{\delta})$, with nearly-optimal search time,

$O(m + (occ + 1)\log^{\epsilon} n)$, which is built in $O(n)$ expected time. This closes a question open in those previous works.

Our result could be enhanced in various ways, as done in the past with γ-bounded indexes [3]. For example, is it possible to search in optimal $O(m + occ)$ time within $O(\delta \log \frac{n}{\delta} \log^{\epsilon} n)$ space? Can we count the number of pattern occurrences in $O(m + \log^{2+\epsilon} n)$ time within our optimal space, or in $O(m)$ time within $O(\delta \log \frac{n}{\delta} \log n)$ space? We believe the answer to all those questions is affirmative and plan to answer them in the extended version of this article.

References

1. Batu, T., Sahinalp, S.C.: Locally consistent parsing and applications to approximate string comparisons. In: 9th International Conference on Developments in Language Theory. LNCS, vol. 3572, pp. 22–35 (2005). https://doi.org/10.1007/11505877_3
2. 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. SIAM (2020). https://doi.org/10.1137/1.9781611975994.37
3. Christiansen, A.R., Ettienne, M.B., Kociumaka, T., Navarro, G., Prezza, N.: Optimal-time dictionary-compressed indexes. ACM Transactions on Algorithms **17**(1), 8:1–8:39 (2021). https://doi.org/10.1145/3426473
4. 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
5. 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
6. 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
7. 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
8. 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
9. 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
10. Kociumaka, T., Navarro, G., Prezza, N.: Towards a definitive measure of repetitiveness. In: 14th Latin American Symposium on Theoretical Informatics, LATIN 2020. LNCS, vol. 12118, pp. 207–219 (2020). https://doi.org/10.1007/978-3-030-61792-9_17
11. Kociumaka, T., Navarro, G., Prezza, N.: Towards a definitive compressibility measure for repetitive sequences (Oct 2021), https://arxiv.org/pdf/1910.02151
12. Kociumaka, T., Radoszewski, J., Rytter, W., Waleń, T.: Internal pattern matching queries in text and applications (2021), Unpublished manuscript

13. 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
14. 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
15. 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
16. Navarro, G.: Indexing highly repetitive string collections, part I: Repetitiveness measures. ACM Computing Surveys **54**(2), 29:1–29:31 (2021). https://doi.org/10.1145/3434399
17. Navarro, G.: Indexing highly repetitive string collections, part II: Compressed indexes. ACM Computing Surveys **54**(2), 26:1–26:32 (2021). https://doi.org/10.1145/3432999
18. 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
19. 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. Springer (1994). https://doi.org/10.1007/3-540-57811-0_3
20. 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), e1002195 (2015). https://doi.org/10.1371/journal.pbio.1002195