Counting on General Run-Length Grammars

Gonzalo Navarro
Center for Biotechnology and Bioengineering (CeBiB)
Department of Computer Science, University of Chile, Chile

Alejandro Pacheco
Center for Biotechnology and Bioengineering (CeBiB)
Department of Computer Science, University of Chile, Chile

Abstract

We introduce a data structure for counting pattern occurrences in texts compressed with any run-length context-free grammar. Our structure uses space proportional to the grammar size and counts the occurrences of a pattern of length m in a text of length n in time $O(m \log^{2+\epsilon} n)$, for any constant $\epsilon > 0$ chosen at indexing time. This is the first solution to an open problem posed by Christiansen et al. [ACM TALG 2020] and enhances our abilities for computation over compressed data; we give an example application.

2012 ACM Subject Classification Theory of computation → Data structures design and analysis

Keywords and phrases Grammar-based indexing; Run-length context-free grammars, Counting pattern occurrences; Periods in strings.

Digital Object Identifier 10.4230/LIPIcs.CPM.2025.

Funding *Gonzalo Navarro*: Funded by Basal Funds FB0001, Mideplan, Chile, and Fondecyt Grant 1-230755, Chile.
Alejandro Pacheco: Funded by Basal Funds FB0001, Mideplan, Chile, Fondecyt Grant 1-230755, Chile, and ANID/Scholarship Program/DOCTORADO BECAS CHILE/2018-21180760.

1 Introduction

Context-free grammars (CFGs) have proven to be an elegant and efficient model for data compression. The idea of grammar-based compression [51, 29] is, given a text $T[1..n]$, to construct a context-free grammar G of size g that only generates T. One can then store G instead of T, which achieves compression if $g \ll n$. Compared to more powerful compression methods like Lempel-Ziv [35], grammar compression offers efficient direct access to arbitrary snippets of T without the need of full decompression [49, 3]. This has been extended to offering indexed searches (i.e., in time $o(n)$) for the occurrences of string patterns in T [8, 16, 10, 7, 40], as well as more complex computations over the compressed sequence [32, 21, 18, 19, 41, 28]. Since finding the smallest grammar G representing a given text T is NP-hard [49, 5], many algorithms have been proposed to find small grammars for a given text [34, 49, 46, 50, 36, 23, 24]. Grammar compression is particularly effective when handling repetitive texts; indeed, the size g^* of the smallest grammar representing T is used as a measure of its repetitiveness [39].

Nishimoto et al. [47] proposed enhancing CFGs with "run-length rules" to improve the compression of repetitive strings. These run-length rules have the form $A \rightarrow B^s$, where B is a terminal or a non-terminal symbol and $s \geq 2$ is an integer. CFGs that may use run-length rules are called run-length context-free grammars (RLCFGs). Because CFGs are RLCFGs, the size g^*_{rl} of the smallest RLCFG generating T always satisfies $g^*_{rl} \leq g^*$, and it can be $g^*_{rl} = o(g^*)$ in text families as simple as $T = a^n$, where $g^*_{rl} = O(1)$ and $g^* = \Theta(\log n)$.

The use of run-length rules has become essential to produce grammars with size guarantees and convenient regularities that speed up indexed searches and other computations [32, 21, 18, 7, 28, 30]. The progress made in indexing texts with CFGs has been extended to RLCFGs, reaching the same status in most cases. These functionalities include extracting substrings, computing substring summaries, and locating all the occurrences of a pattern string [7, App. A]. It has also been shown that RLCFGs can be balanced [42] in the same way as CFGs [19], which simplifies many compressed computations on RLCFGs.

Interestingly, *counting*, that is, determining how many times a pattern occurs in the text without spending the time to list those occurrences, can be done efficiently on CFGs, but not so far on RLCFGs. Counting is useful in various fields, such as pattern discovery and ranked retrieval, for example to help determine the frequency or relevance of a pattern in the texts of a collection [37].

Navarro [44] showed how to count the occurrences of a pattern $P[1..m]$ in $T[1..n]$ in $O(m^2 + m\log^{2+\epsilon} n)$ time using $O(g)$ space if a CFG of size g represents T, for any constant $\epsilon > 0$ chosen at indexing time. Christiansen et al. improved this time to $O(m\log^{2+\epsilon} n)$ by using more recent underlying data structures for tries. Christiansen et al. [7] and Kociumaka et al. [30] extended the result to *particular* RLCFGs, even achieving optimal $O(m)$ time by using additional space, but could not extend their mechanism to general RLCFGs. Their paper [7] finishes, referring to counting, with "However, this holds only for CFGs. Run-length rules introduce significant challenges [...] An interesting open problem is to generalize this solution to arbitrary RLCFGs."

In this paper we give the first solution to this open problem, by introducing an index that counts the occurrences of a pattern $P[1..m]$ in a text $T[1..n]$ represented by a RLCFG of size g_{rl}. Our index uses $O(g_{rl})$ space and answers queries in time $O(m\log^{2+\epsilon} n)$ for any constant $\epsilon > 0$ chosen at indexing time. This is the same time complexity that holds for CFGs, which puts on par our capabilities to handle RLCFGs and CFGs on all the considered functionalities. As an example of our new capabilities, we show how a recent result on finding the maximal exact matches of P using CFGs [45] can now run on RLCFGs.

While our solution builds on the ideas developed for CFGs and particular RLCFGs [44, 7, 30], arbitrary RLCFGs lack crucial structure that holds in those particular cases, namely that if there exists a run-length rule $A \rightarrow B^s$, then the period [11] of the string represented by A is the length of that of B. We show, however, that the general case still retains some structure relating the shortest periods of P and the string represented by A. We exploit this relation to develop a solution that, while considerably more complex than that for those particular cases, retains the same theoretical guarantees obtained for CFGs.

2 Basic Concepts

2.1 Strings

A *string* $S[1\,..\,n] = S[1] \cdot S[2] \cdots S[n]$ is a sequence of symbols, where each symbol belongs to a finite ordered set of integers called an *alphabet* $\Sigma = \{1, 2, \dots, \sigma\}$. The *length* of S is denoted by $|S| = n$. We denote with ε the empty string, where $|\varepsilon| = 0$. A *substring* of S is $S[i\,..\,j] = S[i] \cdot S[i+1] \cdots S[j]$ (which is ε if $i > j$). A *prefix (suffix)* is a substring of the form $S[\,..\,j] = S[1\,..\,j]$ ($S[j\,..\,] = S[j\,..\,n]$); we also say that $S[\,..\,j]$ ($S[j\,..\,]$) *prefixes* (*suffixes*) S. We write $S \sqsubseteq S'$ if S prefixes S', and $S \sqsubset S'$ if in addition $S \neq S'$ (S strictly prefixes S').

We denote with $S \cdot S'$ the *concatenation* of S and S'. A *power* $t \in \mathbb{N}$ of a string S, written S^t, is the concatenation of t copies of S. The *reverse* string of $S[1\,..\,n] = S[1] \cdot S[2] \cdots S[n]$ refers to $S[1\,..\,n]^{\mathrm{rev}} = S[n] \cdot S[n-1] \cdots S[1]$. We also use the term *text* to refer to a string.

2.2 Periods of strings

Periods of strings [11] are crucial in this paper. We recall their definition(s) and a key property, the renowned Periodicity Lemma.

▶ **Definition 1.** *A string $S[1\,..\,n]$ has a* period $1 \le p \le n$ *if, equivalently,*

1. *it consists of $\lfloor n/p \rfloor$ consecutive copies of $S[1\,..\,p]$ plus a (possibly empty) prefix of $S[1\,..\,p]$, that is, $S = (S[1\,..\,p]^{\lceil n/p \rceil})[1\,..\,n]$; or*
2. *$S[1\,..\,n-p] = S[p+1\,..\,n]$; or*
3. *$S[i+p] = S[i]$ for all $1 \le i \le n-p$.*

We also say that p is a period of S. We define $p(S)$ as the shortest period of a non-empty string S and say S is periodic *if $p(S) \le n/2$.*

▶ **Lemma 2** ([14]). *If p and p' are periods of S and $|S| \ge p + p' - \gcd(p, p')$, then $\gcd(p, p')$ is a period of S. Thus, $p(S)$ divides all other periods $p \le |S|/2$ of S.*

2.3 Karp-Rabin signatures

Karp–Rabin [26] fingerprinting assigns a function $k(S) = (\sum_{i=1}^{m} S[i] \cdot c^{i-1}) \bmod \mu$ to the string $S[1\,..\,m]$, where c is a suitable integer and μ a prime number. Bille et al. [4] showed how to build, in $O(n \log n)$ expected time, a *Karp–Rabin signature* $\kappa(S)$ built from a pair of Karp–Rabin functions, which has no collisions between substrings S of $T[1\,..\,n]$. We always assume those kind of signatures in this paper.

A well-known property is that we can compute the functions $k(S[\,..\,j])$ for all the prefixes $S[\,..\,j] \sqsubseteq S$ in time $O(m)$, and then obtain any function $k(S[i\,..\,j])$ (and, consequently, any signature $\kappa(S[i\,..\,j])$) in constant time by using arithmetic operations.

2.4 Range summary queries on grids

A discrete grid of r rows and c columns stores points at integer coordinates (x, y), with $1 \leq x \leq c$ and $1 \leq y \leq r$. Grids with m points can be stored in $O(m)$ space, so that some *summary* queries are performed on *orthogonal ranges* of the grid. In particular, one can associate an integer with each point, and then, given an orthogonal range $[x_1, x_2] \times [y_1, y_2]$, compute the *sum* of all the integers associated with the points in that range. Chazelle [6] showed how to run that query in time $O(\log^{2+\epsilon} m)$, for any constant $\epsilon > 0$, in $O(m)$ space, which works for any semigroup. Navarro [44] describes a simpler solution for groups.

2.5 Grammar compression and parse trees

A *context-free grammar (CFG)* $G = (V, \Sigma, R, S)$ is a language generation model consisting of a finite set of nonterminal symbols V and a finite set of terminal symbols Σ, disjoint from V. The set R contains a finite set of production rules $A \to \alpha$, where A is a nonterminal symbol and α is a string of terminal and nonterminal symbols. The language generation process starts from a sequence formed by just the nonterminal $S \in V$ and, iteratively, chooses a rule $A \to \alpha$ and replaces an occurrence of A in the sequence by α, until the sequence contains only terminals. The *size* of the grammar, $g = |G|$, is the sum of the lengths of the right-hand sides of the rules, $g = \sum_{A \to \alpha \in R} |\alpha|$. Given a string T, we can build a CFG G that generates only T. Then, especially if T is repetitive, G is a compressed representation of T. The *expansion* $\exp(A)$ of a nonterminal A is the string generated by A, for instance $\exp(S) = T$; for terminals a we also say $\exp(a) = a$. We use $|A| = |\exp(A)|$ and $p(A) = p(\exp(A))$.

The *parse tree* of a grammar is an ordinal labeled tree where the root is labeled with the initial symbol S, the leaves are labeled with terminal symbols, and internal nodes are labeled with nonterminals. If $A \to \alpha_1 \cdots \alpha_t$, with $\alpha_i \in V \cup \Sigma$, then a node v labeled A has t children labeled, left to right, $\alpha_1, \ldots, \alpha_t$. A more compact version of the parse tree is the *grammar tree*, which is obtained by pruning the parse tree such that only one internal node labeled A is kept for each nonterminal A, while the rest become leaves. Unlike the parse tree, the grammar tree of G has only $g + 1$ nodes. Consequently, the text T can be divided into at most g substrings, called *phrases*, each being the expansion of a grammar tree leaf. The starting phrase positions constitute a *string attractor* of the text [27]. Therefore, all text substrings of length more than 1 have at least one occurrence that crosses a phrase boundary.

2.6 Run-length grammars

Run-length CFGs (RLCFGs) [47] extend CFGs by allowing in R rules of the form $A \to \beta^s$, where $s \geq 2$ is an integer and β is a string of terminals and nonterminals. These rules are equivalent to rules $A \to \beta \cdots \beta$ with s repetitions of β. However, the length of the right-hand side of the rule A is defined as $|\beta| + 1$, not $s \cdot |\beta|$. To simplify, we will only allow run-length rules of the form $A \to B^s$, where B is a single terminal or nonterminal; this does not increase the asymptotic grammar size because we can rewrite $A \to B^s$ and $B \to \beta$ for a fresh B.

RLCFGs are never larger than general CFGs, and they can be asymptotically smaller. For example, the size g^*_{rl} of the smallest RLCFG that generates T is in $O(\delta \log \frac{n \log |\Sigma|}{\delta \log n})$, where δ is a measure of repetitiveness based on substring complexity [48, 31], but such a bound does not always hold for the size g^* of the smallest grammar. The maximum stretch between g^* and g^*_{rl} is $O(\log n)$, as we can replace each rule $A \to B^s$ by $O(\log s)$ CFG rules.

We denote the size of an RLCFG G as $g_{rl} = |G|$. To maintain the invariant that the grammar tree has $g_{rl} + 1$ nodes, we represent rules $A \to B^s$ as a node labeled A with two children: the first is B and the second is a special leaf $B^{[s-1]}$, denoting $s - 1$ repetitions of B.

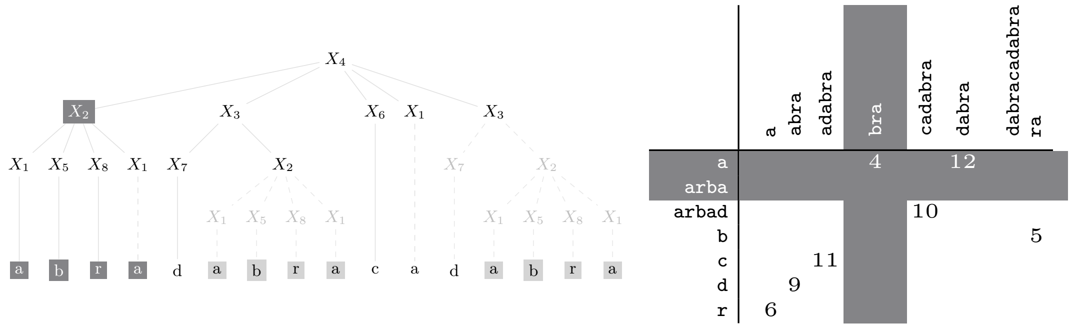

Figure 1 On the left, a grammar tree for $T =$ `abradabracadabra` (with straight solid edges), so $\exp(X_4) = T$. Dashed edges were removed from the parse tree. The only primary occurrence of $P =$ `abra` in T is marked with dark gray on the bottom; the secondary ones are in light gray. On the right, the grid used for searching primary occurrences. Gray stripes indicate the search ranges corresponding to the partition $P = R \mid Q$, where $R =$ `a` and $Q =$ `bra`. The value 4 stored in the resulting cell is the preorder of the child X_5 of the locus node X_2 where Q starts.

3 Grammar Indexing for Locating

A *grammar index* represents a text $T[1\,..\,n]$ using a grammar G that generates only T. As opposed to mere compression, the index supports three primary pattern-matching queries: *locate* (returning all positions of a pattern in the text), *count* (returning the number of times a pattern appears in the text), and *extract* (extracting any desired substring of T). In order to locate, grammar indexes identify "initial" pattern occurrences and then track their "copies" throughout the text. The former are the *primary occurrences*, defined as those that cross phrase boundaries, and the latter are the *secondary occurrences*, which are confined to a single phrase. This approach [25] forms the basis of most grammar indexes [8, 9, 10] and related ones [16, 33, 12, 17, 13, 2, 43, 52], which first locate the primary occurrences and then derive their secondary occurrences through the grammar tree.

As mentioned in Section 2.5, the grammar tree leaves cut the text into phrases. In order to report each primary occurrence of a pattern $P[1\,..\,m]$ exactly once, let v be the lowest common ancestor of the first and last leaves the occurrence spans; v is called the *locus node* of the occurrence. Let v have t children and the first leaf that covers the occurrence descend from the ith child of v. If v represents $A \to \alpha_1 \cdots \alpha_t$, it follows that $\exp(\alpha_i)$ finishes with a pattern prefix $R = P[1\,..\,q]$ and that $\exp(\alpha_{i+1}) \cdots \exp(\alpha_t)$ starts with the suffix $Q = P[q+1\,..\,m]$. We will denote such *cuts* as $P = R \mid Q$. The alignment of $R \mid Q$ within $\exp(\alpha_i) \mid \exp(\alpha_{i+1}) \cdots \exp(\alpha_t)$ is the only possible one for that primary occurrence.

Following the original scheme [25], grammar indexing builds two sets of strings, $\mathcal{X}$ and $\mathcal{Y}$, to find primary occurrences [8, 9, 10]. For each grammar rule $A \to \alpha_1 \cdots \alpha_t$, the set $\mathcal{X}$ contains all the reverse expansions of the children of A, $\exp(\alpha_i)^{\mathrm{rev}}$, and $\mathcal{Y}$ contains all the expansions of the nonempty rule suffixes, $\exp(\alpha_{i+1}) \cdots \exp(\alpha_t)$. Both sets are sorted lexicographically and placed on a grid with (less than) g points, $t-1$ for each rule $A \to \alpha_1 \cdots \alpha_t$. Given a pattern $P[1\,..\,m]$, for each cut $P = R \mid Q$, we first find the lexicographic ranges $[s_x, e_x]$ of R^{rev} in $\mathcal{X}$ and $[s_y, e_y]$ of Q in $\mathcal{Y}$. Each point $(x, y) \in [s_x, e_x] \times [s_y, e_y]$ represents a primary occurrence of P. Grid points are augmented with their locus node v and offset $|\exp(\alpha_1) \cdots \exp(\alpha_i)|$. The cut-based approach naturally extends to the case $m = 1$ by allowing empty prefixes, that is, cuts of the form $P = \varepsilon \mid P[1]$. We then search for suffixes matching $P[1]$ in $\mathcal{Y}$, combining them with all rows in $\mathcal{X}$ to retrieve all primary occurrences of the character.

Once we identify the locus node v (with label A) of a primary occurrence, every other mention of A or its ancestors in the grammar tree, and recursively, of the ancestors of those mentions, yields a secondary occurrence of P. Those are efficiently tracked and reported [9, 10, 7]. An important *consistency* observation for counting is that the amount of secondary occurrences triggered by each primary occurrence is fixed. See Figure 1.

The original approach [9, 10] spends time $O(m^2)$ to find the ranges $[s_x, e_x]$ and $[s_y, e_y]$ for the $m-1$ cuts of P; this was later improved to $O(m \log n)$ [7]. Each primary occurrence found in the grid ranges takes time $O(\log^\epsilon g)$ using geometric data structures, whereas each secondary occurrence requires $O(1)$ time. Overall, the *occ* occurrences of P in T are listed in time $O(m \log n + occ \log^\epsilon g)$.

To generalize this solution to RLCFGs [7, App. A.4], rules $A \to B^s$ are added as a point $(x, y) = (\exp(B)^{\text{rev}}, \exp(B)^{s-1})$ in the grid. This suffices to capture every primary occurrence of the corresponding rule $A \to B \cdots B$: If there are primary occurrences with the cut $P = R \mid Q$ in $B \cdots B$, then one is aligned with the first phrase boundary, $\exp(B) \mid \exp(B)^{s-1}$. Precisely, there is space to place Q right after the first $t = s - \lceil |Q|/|B| \rceil$ phrase boundaries. When the point (x, y) is retrieved for a given cut, then, t primary occurrences are declared with offsets $|B| - |R|$, $2|B| - |R|$, ..., $t|B| - |R|$ within $\exp(A)$. The amount of secondary occurrences triggered by each such primary occurrence still depends only on A.

4 Counting with Grammars

Navarro [44] obtained the first result in counting the number of occurrences of a pattern $P[1\,..\,m]$ in a text $T[1\,..\,n]$ represented by a CFG of size g, within time $O(m^2 + m \log^{2+\epsilon} g)$, for any constant $\epsilon > 0$, and using $O(g)$ space. His method relies on the *consistency* observation above, which allows enhancing the grid described in Section 3 with the number $c(A)$ of (primary and) secondary occurrences associated with each point. At query time, for each pattern cut, one sums the number of occurrences in the corresponding grid range using the technique mentioned in Section 2.4. The final complexity is obtained by aggregating over all $m-1$ cuts of P and considering the $O(m^2)$ time required to identify all the ranges. Christiansen et al. [7, Thm. A.5] later improved this time to just $O(m \log n + m \log^{2+\epsilon} g)$, by using more modern techniques to find the grid range of all cuts of P.

Christiansen et al. [7] also presented a method to count in $O(m + \log^{2+\epsilon} n)$ time on a *particular* RLCFG of size $g_{rl} = O(\gamma \log(n/\gamma))$, where γ is the size of the smallest string attractor [27] of T. They also show that by increasing the space to $O(\gamma \log(n/\gamma) \log^\epsilon n)$ one can reach the optimal counting time, $O(m)$. The grammar properties allow reducing the number of cuts of P to check to $O(\log m)$, instead of the $m-1$ cuts used on general RLCFGs.

Christiansen et al. build on the same idea of enhancing the grid with the number of secondary occurrences, but the process is considerably more complex on RLCFGs, because the consistency property exploited by Navarro [44] does not hold on run-length rules $A \to B^s$: the number of occurrences triggered by a primary occurrence with cut $P = R \mid Q$ found from the point $(\exp(B)^{\text{rev}}, \exp(B)^{s-1})$ depends on s, $|B|$, and $|Q|$. Their counting approach relies on another property that is specific of their RLCFG [7, Lem. 7.2]:

▶ **Property 1.** *For every run-length rule $A \to B^s$, the shortest period of* $\exp(A)$ *is* $|B|$.

This property facilitates the division of the counting process into two cases. For each run-length rule $A \to B^s$, they introduce two points, $(x, y') = (\exp(B)^{\text{rev}}, \exp(B))$ and $(x, y'') = (\exp(B)^{\text{rev}}, \exp(B)^2)$, in the grid. These points are associated with the values $c(A)$ and $(s-2) \cdot c(A)$, respectively. The counting process is as follows: for a cut $P = R \mid Q$ where R is a suffix of $\exp(B)$, if $Q \sqsubseteq \exp(B)$, then it will be counted $c(A) + (s-2) \cdot c(A) = (s-1) \cdot c(A)$

times, as both points will be within the search range. If Q instead exceeds $\exp(B)$, but still $Q \sqsubseteq \exp(B)^2$, then it will be counted $(s-2) \cdot c(A)$ times, solely by point (x, y''). Finally if Q exceeds $\exp(B)^2$, then Q is periodic (with $p(Q) = |B|$).

They handle that remaining case as follows. Given a cut $P = R \mid Q$ and the period $p = p(Q) = |B|$, where $|Q| > 2p$, the number of primary occurrences of this cut inside rule $A \to B^s$ is $s - \lceil |Q|/p \rceil$ (cf. the end of Section 3). Let D be the set of rules $A \to B^s$ such that R is a suffix of $\exp(B)$ and Q is a prefix of $\exp(B)^{s-1}$, that is, those within the grid range of the cut, and $c(A)$ the number of (primary and secondary) occurrences of A. Then, the number of occurrences triggered by the primary occurrences found within symbols in D for this cut is

$$\sum_{A \to B^s \in D} c(A) \cdot s - c(A) \cdot \lceil |Q|/p \rceil. \tag{1}$$

For each run-length rule $A \to B^s$, they compute a Karp–Rabin signature (Section 2.3) $\kappa(\exp(B))$ and store it in a perfect hash table [15, 1], associated with values

$$C(B, s) = \sum \{c(A) : A \to B^{s'}, s' \geq s\},$$
$$C'(B, s) = \sum \{s' \cdot c(A) : A \to B^{s'}, s' \geq s\}.$$

Additionally, for each such B, the authors store the set $s(B) = \{s : A \to B^s\}$.

At query time, they calculate the shortest period $p = p(P)$. For each cut $P = R \mid Q$, Q is periodic if $|Q| > 2p$. If so, they compute $k = \kappa(Q[1\,..\,p])$, and if there is an entry B associated with k in the hash table, they add to the number of occurrences found up to then

$$C'(B, s_{min}) - C(B, s_{min}) \cdot \lceil |Q|/p \rceil, \tag{2}$$

where $s_{min} = \min\{s \in s(B), (s-1) \cdot |B| \geq |Q|\}$ is computed using exponential search over $s(B)$ in $O(\log m)$ time. Note that they exploit the fact that *the number of repetitions to subtract*, $\lceil |Q|/p \rceil$, depends only on $p = |B|$, and not on the exponent s of rules $A \to B^s$.

Since fingerprints $\kappa(\pi)$ are collision-free on substrings of T, and the nonterminals in their particular RLSLP produce distinct expansions, each valid fingerprint $\kappa(Q[1\,..\,p])$ corresponds to at most one nonterminal B. This guarantees that, if a match is found in the hash table, it uniquely identifies a single candidate B. Further, they show how to filter out false positives for prefixes of Q that do not occur in the set [7, Lem. 6.5].

The total counting time, on a grammar of size g_{rl}, is $O(m \log n + m \log^{2+\epsilon} g_{rl})$. In their grammar, the number of cuts to consider is $O(\log m)$, which allows reducing the cost of computing the grid ranges to $O(m)$. The signatures of all substrings of P are also computed in $O(m)$ time, as mentioned in Section 2.3. Considering the grid searches, the total cost for counting the pattern occurrences drops to $O(m + \log^{2+\epsilon} g_{rl}) \subseteq O(m + \log^{2+\epsilon} n)$ [7, Sec. 7].

Recently, Kociumaka et al. [30] employed this same approach to count the occurrences of a pattern in a smaller RLCFG that uses $O(\delta \log \frac{n \log |\Sigma|}{\delta \log n})$ space, where $\delta \leq \gamma$. They demonstrated that the RLCFG they produce satisfies Property 1 [7, Lem. 7.2], which is necessary to apply the described scheme.

5 Our Solution

We now describe a solution to count the occurrences in arbitrary RLCFGs, where the convenient Property 1 used in the literature may not hold. We start with a simple observation.

▶ **Lemma 3.** *Let $A \to B^s$ be a rule in a RLCFG. Then $p(A)$ divides $|B|$.*

Proof. Clearly $|B|$ is a period of $\exp(A)$ because $\exp(A) = \exp(B)^s$. By Lemma 2, then, since $|B| \leq |A|/2$, $p(A)$ divides $|B|$. ◀

Some parts of our solution make use of the shortest period of $\exp(A)$. We now define some related notation.

▶ **Definition 4.** *Given a rule $A \to B^s$ with $s \geq 2$, let $p = p(A)$ (which divides $|B|$ by Lemma 3). The corresponding* transformed rule *is $A \to \hat{B}^{\hat{s}}$, where $\hat{B}$ is a new nonterminal such that $\exp(\hat{B}) = \exp(A)[1\,..\,p]$, and $\hat{s} = s \cdot (|B|/p)$.*

There seems to be no way to just transform all run-length rules (which would satisfy Property 1, $p(A) = |\hat{B}|$) without blowing up the RLCFG size by a logarithmic factor. We will use another approach instead. We classify the rules into two categories.

▶ **Definition 5.** *Given a rule $A \to B^s$ with $s \geq 2$, we say that A is of type-E (for Equal) if $p(A) = |\hat{B}| = |B|$; otherwise, $p(A) = |\hat{B}| < |B|$ and we say that A is of type-L (for Less).*

We build on Navarro's solution [44] for counting on CFGs, which uses an enhanced grid where points count all the occurrences they trigger. The grid ranges are found with the more recent technique [7] that takes $O(m \log n)$ time. Further, we treat type-E rules exactly as Christiansen et al. [7] handle the run-length rules in their specific RLCFGs, as described in Section 4. This is possible because type-E rules, by definition, satisfy Property 1. Their method, however, assumes that no two symbols $B \neq B'$ have the same expansion. To relax this assumption, symbols B with the same expansion should collectively contribute to the same entries of $C(\cdot, s)$ and $C'(\cdot, s)$. We thus index those tables using $\kappa(\exp(B))$ rather than B, and for simplicity write $C(\pi, s)$, $C'(\pi, s)$, and $s(\pi)$, where $\pi = \exp(B)$. Further, the time to filter our false positives using their Lemma 6.5 [7] is $O(m \log n)$ because we must explore all the $m-1$ cuts of P.

Since each primary occurrence is found in exactly one rule, we can decompose the process of counting by adding up the occurrences found inside type-E and type-L rules. We are then left with the more complicated problem of counting occurrences found from type-L rules. We start with another observation.

▶ **Observation 6.** *If $A \to B^s$ is a type-L rule, then $|B| \geq 2|\hat{B}|$*

Proof. If A is a type-L rule then $p(A) = |\hat{B}| < |B|$. In addition, by Lemma 3, $|\hat{B}|$ divides $|B|$. Therefore $|B| \geq 2|\hat{B}|$ ◀

For type-L rules, we will generalize the strategy of Section 4: the cases where $|Q| \leq 2|\hat{B}|$ will be handled by adding points to the enhanced grid; in the other cases we will use new data structures that exploit the fact (to be proved) that Q is periodic. Note that each cut $P = R \mid Q$ may correspond to different cases for different run-length rules, so our technique will consider all the cases for each cut. Although the primary occurrences within a rule $A \to B^s$ will still be defined as those that cross boundaries of B, we will find them by aligning (all the possible) cuts $P = R \mid Q$ with the boundaries of the nonterminals $\hat{B}$ of the transformed rules $A \to \hat{B}^{\hat{s}}$. The following definition will help us show how we capture every primary occurrence exactly once.

▶ **Definition 7.** *The* alignment *of a primary occurrence x found with cut $P = R \mid Q$ inside the type-L rule $A \to B^s$ is $align(x) = 1 + ((|R| - 1) \bmod |\hat{B}|)$.*

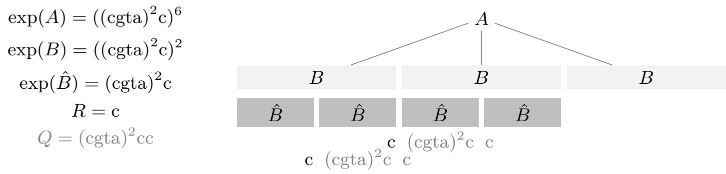

Figure 2 We show the occurrences captured by the point $(x_p, y''_p) = (\exp(\hat{B}), \exp(\hat{B})^2)$. Note how the occurrence in the first row is correctly captured by (x_p, y''_p), whereas that in the second row is not captured by any point. Consequently, the first row is effectively counted twice. Given that the point (x_p, y''_p) is assigned a weight of $2 \cdot (s-1) \cdot c(A)$, the total number of occurrences is $4 \cdot c(A)$.

The definition is sound because every primary occurrence is found using exactly one cut $P = R \mid Q$. Note that *align* $\in [1\,..\,|\hat{B}|]$ is the distance from the starting position of an occurrence, within $\exp(A)$, to the start of the next copy of $\exp(\hat{B})$. We will explore all the possible cuts of P, but each rule $A \rightarrow B^s$ will be probed only with the cuts where $1 \le |R| \le |\hat{B}|$. From those cuts, all the corresponding primary occurrences aligned with the $\hat{s}-1$ boundaries between copies of $\hat{B}$ (i.e., with the same alignment, $|R|$) will be captured.

5.1 Case $|Q| \le 2|\hat{B}|$

To capture the primary occurrences with cut $P = R \mid Q$ inside type-L rules $A \rightarrow B^s$ where $|Q| \le 2|\hat{B}|$, we will incorporate the points $(x_p, y'_p) = (\exp(\hat{B})^{\text{rev}}, \exp(\hat{B}))$ and $(x_p, y''_p) = (\exp(\hat{B})^{\text{rev}}, \exp(\hat{B})^2)$ into the enhanced grid outlined in Sections 3 and 4, assigning the values $-(s-1) \cdot c(A)$ and $2 \cdot (s-1) \cdot c(A)$ to each, respectively. The point (x_p, y'_p) will capture the occurrences where $|R|, |Q| \le |\hat{B}|$. Note that these occurrences will also find the point (x_p, y''_p), so the final result will be $(2-1) \cdot (s-1) \cdot c(A) = (s-1) \cdot c(A)$.

The point (x_p, y''_p) will also account for the primary occurrences where $|R| \le |\hat{B}|$ and $|\hat{B}| < |Q| \le 2|\hat{B}|$. Observation 6 establishes that $|B| \ge 2|\hat{B}|$, so for each such primary occurrence of cut $R \mid Q$, with offset j in $\exp(A)$, there is a second primary occurrence at $j - |\hat{B}|$ with cut $P = R' \mid Q'$, where $|\hat{B}| < |R'| = |R| + |\hat{B}| \le 2|\hat{B}|$ and $|Q'| = |Q| - |\hat{B}| \le |\hat{B}|$. This second cut will not be captured by the points we have inserted because $|R'| > |\hat{B}|$. The other occurrences where P matches to the left of $j - |\hat{B}|$ fall within B (and thus are not primary), because we already have $|Q'| \le |\hat{B}|$ in this second occurrence. Thus, for each of the s copies of B (save the last), we will have two primary occurrences. This yields a total of $2 \cdot (s-1) \cdot c(A)$ occurrences, which are properly counted in the points (x_p, y''_p). See Figure 2.

5.2 Case $|Q| > 2|\hat{B}|$

We first show that, for Q to be longer than $2|\hat{B}|$ in some run-length rule, P must be periodic.

▶ **Lemma 8.** *Let P, with $p = p(P)$, have a primary occurrence with cut $P = R \mid Q$ in the rule $A \rightarrow B^s$, with $p(A) = |\hat{B}|$ and $|Q| > 2|\hat{B}|$. Then it holds that $p = p(A)$.*

Proof. Since $|P| \ge |\hat{B}|$ and P is contained within $\exp(A) = \exp(\hat{B})^{\hat{s}}$, by branch 3 of Definition 1, $|\hat{B}|$ must be a period of P. Thus, $p = p(P) \le |\hat{B}|$. Suppose, for contradiction, that $p < |\hat{B}|$. According to Lemma 2, because $|\hat{B}| \le |Q|/2 \le |P|/2$ is a period of P, it follows that p divides $|\hat{B}|$. Since $\exp(\hat{B})$ is contained in P, again by branch 3 of Definition 1 it follows that $p < |\hat{B}| \le |B|$ is a period of $\exp(B)$, and thus of $\exp(A)$, contradicting the assumption that $p(A) = |\hat{B}|$. Hence, we conclude that $p = |\hat{B}|$. ◀

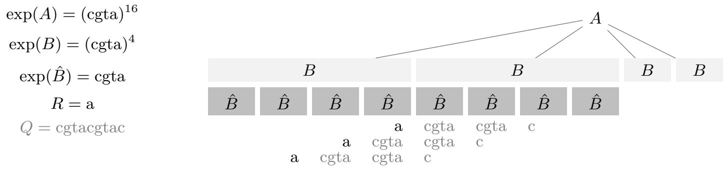

Figure 3 If $2|\hat{B}| < |Q| \leq |B|$, there are $\lceil |Q|/p \rceil$ primary occurrences around the boundary between any two blocks B (we zoom on one) with the cut $P = R \mid Q$. We show the possible alignments of P below the blocks $\hat{B}$. For a rule $A \to B^s$ there are $(s-1)$ boundaries, yielding $(s-1) \cdot \lceil |Q|/p \rceil$ primary occurrences. In this case, $\lceil |Q|/p \rceil = 3$ and $s - 1 = 3$, yielding 9 primary occurrences.

Note that P is then periodic because $p(P) = p(A) = |\hat{B}| < |Q|/2 \leq |P|/2$, and Q is also periodic by branch 3 of Def. 1, because it occurs inside P and $|Q| \geq 2p$.

We distinguish two subcases, depending on whether Q is longer than B or not. If it is, we must ensure that in the alignments we count the occurrence is fully within $\exp(A)$. If it is not, we must ensure that the alignments we count do correspond to primary occurrences (i.e., they cross a border between copies of B).

5.2.1 Case $2|\hat{B}| < |Q| \leq |B|$

To handle this case, we construct a specific data structure based on the period $|\hat{B}|$. The proposed solution is supported by the following lemma.

▶ **Lemma 9.** *Let P, with $p = p(P)$, have a primary occurrence with cut $P = R \mid Q$ in the type-L rule $A \to B^s$, with $p(A) = |\hat{B}|$, $|R| \leq |\hat{B}|$, and $2|\hat{B}| < |Q| \leq |B|$. Then, the number of primary occurrences of P in $\exp(A)$ is $(s-1) \cdot \lceil |Q|/p \rceil$.*

Proof. Since $|R| \leq |\hat{B}|$, R can be aligned at the end of the $|B|/|\hat{B}|$ positions where $\exp(\hat{B})$ starts in $\exp(B)$. No other alignments are possible for the cut $R \mid Q$ because, by Lemma 8, $p = |\hat{B}|$ and another alignment would imply that P aligns with itself with an offset smaller than p, a contradiction by branch 2 of Definition 1.

Those alignments correspond to primary occurrences only if P does not fall completely within $\exp(B)$. The alignments that correspond to primary occurrences are then those where R is aligned at the end of the last $\lceil |Q|/|\hat{B}| \rceil$ ending positions of copies of $\hat{B}$, all of which start within $\exp(B)$ because $|Q| \leq |B|$. This is equivalent to $\lceil |Q|/p \rceil$, as $p = |\hat{B}|$ by Lemma 8. Thus, the number of primary occurrences of P in A is $(s-1) \cdot \lceil |Q|/p \rceil$. See Figure 3. ◀

Based on Lemma 9 we introduce our first period-based data structure. Considering the solution described in Section 4, where Property 1 holds, the challenge with type-L rules $A \to B^s$ (i.e., rules that differ from their transformed version $A \to \hat{B}^{\hat{s}}$) is that the number of alignments with cut $R \mid Q$ inside $\exp(A)$ is $(s-1) \cdot \lceil |Q|/p \rceil$, but $|B|$ does not determine $p = p(A)$. We will instead use $\hat{B}$ to index those nonterminals A.

For each type-L rule $A \to B^s$ ($A \to \hat{B}^{\hat{s}}$ being its transformed version), we compute its signature $\kappa(\exp(\hat{B}))$ (recall Section 2.3) and store it in a perfect hash table H. Each entry in table H, which corresponds to a specific signature $\kappa(\pi)$, will be linked to an array F_π. Each position $F_\pi[i]$ represents a type-L rule $A_i \to B_i^{s_i}$ where $\kappa(\exp(\hat{B}_i)) = \kappa(\pi)$. The rules A_i are sorted in F_π by decreasing lengths $|B_i|$. We also store a field with the cumulative sum

$$F_\pi[i].sum = \sum_{1 \leq j \leq i} (s_j - 1) \cdot c(A_j).$$

Given a pattern $P[1..m]$, we first calculate its shortest period $p = p(P)$. For each cut $P = R \mid Q$ with $1 \le |R| \le \min(p, m-2p-1)$, we compute $\kappa(\pi)$ for $\pi = Q[1..p]$ to identify the corresponding array F_π in H. Note that we only consider the cuts $R \mid Q$ where $|R| \le p$, as this corresponds precisely to $|R| \le |\hat{B}|$ for the rules stored in F_π; note $p = |\pi|$. In addition, the condition $|R| \le m - 2p - 1$ ensures that $|Q| > 2p = 2|\hat{B}|$, thus we are correctly enforcing the condition stated in this subsection and focusing, one by one, on the occurrences x for which each alignment satisfies $align(x) = |R|$. We will find in H every (transformed) rule $A \to \hat{B}^{\hat{s}}$ where $\hat{B} = \pi$, sharing the period p with Q, as well as its prefix $\pi = \exp(B)[1..p] = Q[1..p]$. Once we have obtained the array F_π, we find the largest i such that $|B_i| \ge |Q|$. The number of primary occurrences for the cut $P = R \mid Q$ in type-L rules where $2|\hat{B}| < |Q| \le |B|$ is then $F_\pi[i].sum \cdot \lceil |Q|/p \rceil$.

5.2.2 Case $|Q| > |B|$

Our analysis for the remaining case is grounded on the following lemma.

▶ **Lemma 10.** *Let P, with $p = p(P)$, have a primary occurrence in a type-L rule $A \to B^s$ with cut $P = R \mid Q$, with $|R| \le p$ and $|Q| > |B|$. Then it holds that $p = p(A)$ and $|Q| > 2p$.*

Proof. If A is a type-L rule and P has an occurrence within A such that $|Q| > |B|$, then we have $|Q| > |B| \ge 2|\hat{B}|$ (by Observation 6). Since we can express A as $A \to \hat{B}^{\hat{s}}$, we can similarly use Lemma 8 to conclude that $p = p(A) = |\hat{B}|$; further, $|Q| > 2p$. ◀

Analogously to Lemma 8, Lemma 10 establishes that, when Q is sufficiently long, it holds that $p(P) = p(A)$, so all pertinent rules of the form $A \to B^s$ can be classified according to their minimal period, $p(A)$. This period coincides with $p = p(P)$ when P has an occurrence in a type-L rule such that $|Q| > |B|$. Further, $|Q| > 2p$.

We also need an analogous to Lemma 9 for the case $|Q| > |B|$; this is given next.

▶ **Lemma 11.** *Let P, with $p = p(P)$, have a primary occurrence with cut $P = R \mid Q$ in the type-L rule $A \to B^s$, with $p(A) = |\hat{B}|$, $|R| \le |\hat{B}|$, and $|Q| > |B|$. Then, the number of primary occurrences of P in $\exp(A)$ is $\hat{s} - \lceil |Q|/p \rceil$.*

Proof. Since $|R| \le |\hat{B}|$, R can be aligned at the end of the $\hat{s}$ positions where $\exp(\hat{B})$ starts in $\exp(A)$. By the same argument of the proof of Lemma 9, no other alignments are possible for the cut $R \mid Q$. Unlike in Lemma 9, all those alignments correspond to primary occurrences, because Q is always long enough to exceed B. Also unlike in Lemma 9, Q may exceed A, in which case the occurrence must not be counted in this rule. The alignments that must not be counted are then those where R is aligned at the end of the last $\lceil |Q|/|\hat{B}| \rceil$ ending positions of copies of $\hat{B}$. This is equivalent to $\lceil |Q|/p \rceil$, as $p = |\hat{B}|$ by Lemma 10. Thus, the number of primary occurrences of P in A is $\hat{s} - \lceil |Q|/p \rceil$. See Figure 4. ◀

We then enhance table H, introduced in Section 5.2.1, with a second period-based data structure. Each entry in table H, corresponding to some $\kappa(\pi)$, will additionally store a grid G_π. In this grid, each row represents a type-L rule $A \to B^s$ whose transformed version is $A \to \hat{B}^{\hat{s}}$, that is, such that $\pi = \exp(\hat{B}) = \exp(B)[1..p]$. The rows are sorted by increasing lengths $|B|$ (note $|B| \ge |\pi| = p$ for all B in G_π). The columns represent the different exponents $\hat{s}$ of the transformed rules. The row of rule $A \to B^s$ has then a unique point at

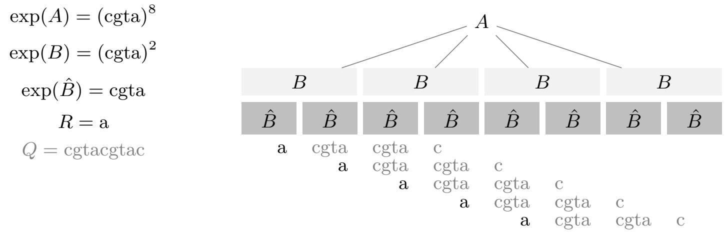

Figure 4 If $|Q| > |B|$, we can compute all occurrences of P around blocks $\hat{B}$ without the risk of any occurrence being fully contained in a block B: the number of primary occurrences of P in $\exp(A)$ is simply $s' - \lceil |Q|/p \rceil$. In this example, with $s' = 8$ and $\lceil |Q|/p \rceil = 3$, there are 5 occurrences.

column $\hat{s}$, and we associate two values with it: $c(A)$ and $c'(A) = \hat{s} \cdot c(A)$. Since no rule appears in more than one grid, the total space for all grids is in $O(g_{rl})$.[1]

Given a pattern $P[1\,..\,m]$, we proceed analogously as explained at the end of Section 5.2.1 in order to identify F_π: We compute $p = p(P)$, and for each cut $P = R \mid Q$ with $1 \le |R| \le \min(p, m - 2p - 1)$, we calculate $\kappa(\pi)$, for $\pi = Q[1\,..\,p]$, to find the corresponding grid G_π in H. On the type-L rules $A \to B^s$, this tries out every possible occurrence x for which $align(x) = |R|$, one by one, from 1 to $|\hat{B}|$. The limit $|R| < m - 2p$ can also be set because, by Lemma 10, it must hold $|Q| > 2|\hat{B}|$ on the rules of G_π we find with the cut $P = R \mid Q$.

We must enforce two conditions on the rules of G_π to consider: (a) $|Q| > |B|$ as corresponds in this subsection, and (b) $\hat{s} - \lceil |Q|/p \rceil \ge 0$, that is, Q fits within $\exp(A)$. The complying rules then contribute $c(A) \cdot (\hat{s} - \lceil |Q|/p \rceil) = c'(A) - c(A) \cdot \lceil |Q|/p \rceil$ by Lemma 11.

To enforce those conditions, we find in G_π the largest row y representing a rule $A \to B^s$ such that $|B| < |Q|$. We also find the smallest column x where $(\hat{s} =)\, x \ge \lceil |Q|/p \rceil$. The set D of rules corresponding to points in the range $[x, n] \times [1, y]$ of the grid is then the set of type-L run-length rules where we have a primary occurrence with $|Q| > |B|$. We aggregate the values $c(A)$ and $c'(A)$ from the range, which yields the correct sum of all the pertinent occurrences (note the analogy with Eqs. (1) and (2)):

$$\left(\sum_{A \to B^s \in D} c'(A) \right) - \left(\sum_{A \to B^s \in D} c(A) \right) \cdot \lceil |Q|/p \rceil \quad = \quad \sum_{A \to B^s \in D} c(A) \cdot \hat{s} - c(A) \cdot \lceil |Q|/p \rceil \,.$$

Figure 5 gives a thorough example.

5.3 The final result

Our structure extends the grid of Section 4, built for non-run-length rules, with one point per run-length rule: those of type-E are handled as described in Section 4 and those of type-L as in Section 5. Thus the structure is of size $O(g_{rl})$ and range queries on the grid take time $O(\log^{2+\epsilon} g_{rl})$. Occurrences on such a grid are counted in time $O(m \log n + m \log^{2+\epsilon} g_{rl})$ [7, Thm. A.5]. This is also the time to count the occurrences in type-E rules for our solution, and those in type-L rules when $|Q| \le 2|B_p|$ (Section 5.1).

For our period-based data structures (Sections 5.2.1 and 5.2.2), we calculate $p(P)$ in $O(m)$ time [11], and compute all prefix signatures of P in $O(m)$ time as well, so that later

[1] We use the grid representation described in Section 2.4, which assumes that the point coordinates lie in rank space. Our grids can be transformed accordingly without affecting the asymptotic space usage or query time.

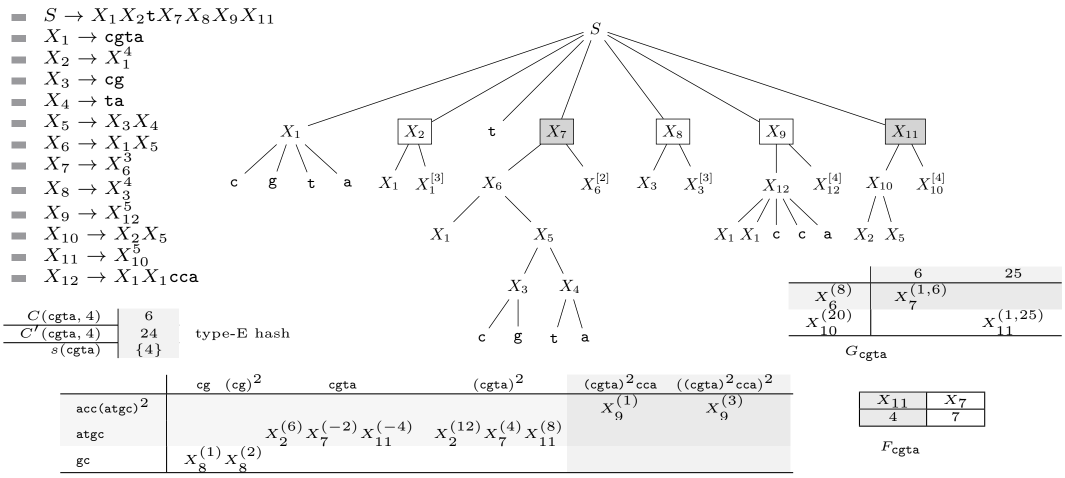

Figure 5 On top, a RLCFG on the left and its grammar tree on the right. Type-E rules are enclosed in white rectangles and Type-L rules in gray rectangles. Below the rules we show the values $C(B, s)$ and $C'(B, s)$ [7] we use to handle the E-type rules (see Section 4); we only show those for $\exp(X_1) = \texttt{cgta}$. On the bottom left we show the points we add to the standard grid. The points for type-E rules are represented as $A^{(c(A))}$ and $A^{((s-2)\cdot c(A))}$ and those for type-L rules as $A^{(-(s-1)\cdot c(A))}$ and $A^{(2\cdot(s-1)\cdot c(A))}$. The bottom right shows the grid G_π and the array F_π for the transformed rules $A \to \hat{B}^{s'}$ where $\hat{B} = \pi = \texttt{cgta}$. In F_π we show the fields $F[i].sum$. In G_π, the row labels show $B^{(|B|)}$ and the column labels show s'; the points show $A^{(c(A),c'(A))}$. Consider the cut $P = \texttt{a} \mid \texttt{cgtacgtac}$, with $p(P) = 4$. We identify 10 occurrences in type-E rules: 4 are found within the rule X_9 using the standard grid, while the remaining 6 are determined via the values of $C(X_1, s)$ and $C'(X_1, s)$. These 6 occurrences specifically arise within $\exp(X_2) = (\texttt{cgta})^4$. Similarly, in the type-L rules, we detect 15 occurrences: 12 occur within the rule X_{11}, identified using the $F_{\texttt{cgta}}$ array, and the remaining 3 arise within $\exp(X_7) = (\texttt{cgta})^6$, captured using the $G_{\texttt{cgta}}$ grid. The final two occurrences of this cut are located using standard CFG rules at $\exp(S)[4\,..\,13]$ $(X_1 \cdot X_2)$ and $\exp(S)[108\,..\,117]$ $(X_9 \cdot X_{11})$. Note that there are 6 additional occurrences: five are obtained using Navarro's solution for counting on CFGs, triggered by a primary occurrence in X_{10}, and the sixth is located using standard CFG rules at $\exp(S)[37\,..\,46]$ $(X_7 \cdot X_8)$. Both groups of occurrences are identified using the cut $P = \texttt{acgtacgta} \mid \texttt{c}$, bringing the total to 33 occurrences of P in the text.

any substring signature is computed in $O(1)$ time (Section 2.3). The limits in the arrays F_π and in the grids G_π can be binary searched in time $O(\log g_{rl})$. The range sums over $c(A)$ and $c'(A)$ take time $O(\log^{2+\epsilon} g_{rl})$. They are repeated for each of the $O(m)$ cuts of P, adding up to time $O(m \log^{2+\epsilon} g_{rl})$. Those are then within the previous time complexities as well.

▶ **Theorem 12.** *Let a RLCFG of size g_{rl} represent a text $T[1\,..\,n]$. Then, for any constant $\epsilon > 0$, we can build in $O(n \log n)$ expected time an index of size $O(g_{rl})$ that counts the number of occurrences of a pattern $P[1\,..\,m]$ in T in time $O(m \log n + m \log^{2+\epsilon} g_{rl}) \subseteq O(m \log^{2+\epsilon} n)$.*

Just as for previous schemes [7, Sec. 6.6], the construction time is dominated by the $O(n \log n)$ expected time to build the collision-free Karp–Rabin functions [4]. Although the construction is randomized, the algorithm is Las-Vegas type and thus it always produces a correct index; query results are always correct and their time is deterministic worst-case. Other construction costs specific of our index are the $O(g_r \log g_r)$ time to build Chazelle's range sums structures [6], and the $O(|A|)$ cost to compute the period $p(A)$ of every run-length rule $A \to B^s$. Those costs sum up to $O(n)$ because the top-level run-length rules in the grammar tree add up to length at most n, and the top-level descendants of A expand at most to $|B| \leq |A|/2$. An easy induction shows that the expansions below A add up to length at most $|A|$, so the total expansion length is at most twice that of the top-level run-length rules.

Space-time tradeoffs

The bulk of the query cost owes to the $O(\log^{2+\epsilon} g_{rl})$ time of the geometric queries. Other space-time tradeoffs are possible. We start with a geometric result of independent interest.

▶ **Lemma 13.** *For any constant $0 < \delta < 1$, we can build in $O(r \log r)$ time a data structure representing r weighted points on an $r \times r$ grid, using space $O(r \log^{1-\delta} r)$, which can sum the weights on any orthohonal range in time $O(\log^{1+\delta} r \log\log r)$. It is also possible to obtain (1) $O(r \log\log r)$ space and $O(\log^2 r \log\log r)$ time and (2) $O(r \log r)$ space and $O(\log r)$ time.*

Proof. Navarro's solution [44, Thm. 3] represents such a grid with a wavelet tree [22] (assuming there is exactly one point per column, but it is easy to reduce the general case to this one). This structure has $\log r$ levels. The r grid points are represented in x-coordinate order in the first level, and their order is progressively shuffled until the last level, which represents the points in y-coordinate order. The coordinates are not represented explicitly; only one bit is used to represent each point at each level, for a total of $O(r \log r)$ bits (which is in $O(r)$ space if measured in words). A two-dimensional query is projected onto $O(\log r)$ ranges along different levels, and the query must sum the weights of the points across all those ranges. To save (space and) time, (only) one cumulative sum is precomputed and stored every $\log r$ consecutive weights at every level, so that in total only $O(r)$ sums are stored overall, and $O(r)$ space is used for those accumulators.

When adding the weights over one range, the sum over most of it is obtained by subtracting two accumulators, and just $O(\log r)$ weights must be explicitly obtained to complete the sum. Those weights are obtained with a structure [6, 38] that takes $O((1/\epsilon) \log^{\epsilon} r)$ time and $O((1/\epsilon) r \log r)$ bits (or $O(r/\epsilon)$ words) of additional space, for any $\epsilon > 0$. Multiplying the $O(\log r)$ ranges to sum, the $O(\log r)$ explicit weights to obtain in each range, and the cost to obtain each weight, we reach the $O(\log^{2+\epsilon} r)$ claimed term [44], using constant ϵ.

To obtain the desired tradeoff, we will set accumulators every $\log^{\delta} r$ values, which yields $O(r \log^{1-\delta} r)$ space. The time will be then $O((1/\epsilon) \log^{1+\delta+\epsilon} r)$. By choosing a non-constant $\epsilon = 1/\log\log r$, the space of the data structure to compute individual weights raises to $O(r \log\log r) \subseteq O(r \log^{1-\delta} r)$, and the time becomes $O(\log^{1+\delta} r \log\log r)$.

Tradeoff (1) is obtained by setting $\delta = 1$, in which case the space $O(r \log\log r)$ of the data structure to compute individual weights dominates. Tradeoff (2) is obtained by setting $\delta = 0$, in which case we do not need at all that data structure: we have all precomputed prefix sums and answer each range sum in constant time, for a total of $O(\log r)$ time.[2] All the variants are built in $O(r \log r)$ time [6]. ◀

By using those grid representations, we obtain tradeoffs in our index.

▶ **Corollary 14.** *Let a RLCFG of size g_{rl} represent a text $T[1\,..\,n]$. Then, for any constant $0 < \delta < 1$, we can build in $O(n \log n)$ expected time an index of size $O(g_{rl} \log^{1-\delta} g_{rl})$ that counts the occurrences of a pattern $P[1\,..\,m]$ in T in time $O(m \log n + m \log^{1+\delta} g_{rl} \log\log g_{rl}) \subseteq O(m \log^{1+\delta} n \log\log n)$. We can also obtain $O(g_{rl} \log\log g_{rl})$ space with time $O(m \log n + m \log^2 g_{rl} \log\log g_{rl}) \subseteq O(m \log^2 n \log\log n)$, and $O(g_{rl} \log g_{rl})$ space with time $O(m \log n)$.*

[2] Chazelle [6] also obtains tradeoff (1) and explores the other spaces, but his time never goes below $\Theta(\log^2 g_{rl})$ because he addresses the more general case of semigroups, with no inverses. Our result is presented for numeric sums, but it can be extended to algebraic groups.

5.4 An application

Recent work [20, 41] shows how to compute the maximal exact matches (MEMs) of $P[1..m]$ in $T[1..n]$, which are the maximal substrings of P that occur in T, in case T is represented with an arbitrary RLCFG. Navarro [45] extends the results to k-MEMs, which are maximal substrings of P that occur at least k times in T. To obtain good time complexities for large enough k, he resorts to counting occurrences of substrings $P[i..j]$ with the grammar. His Thm. 7, however, works only for CFGs, as no efficient counting algorithm existed on RLCFGs. In turn, his Thm. 8 works only for a particular RLCFG. We can now state his result on an arbitrary RLCFG; by his Thm. 11 this also extends to "k-rare MEMs".

▶ **Corollary 15** (cf. [45, Thm. 7]). *Let a RLCFG of size g_{rl} generate only $T[1..n]$. Then, for any constant $\epsilon > 0$, we can build a data structure of size $O(g_{rl})$ that finds the k-MEMs of any given pattern $P[1..m]$, for any $k > 0$ given with P, in time $O(m^2 \log^{2+\epsilon} g_{rl})$.*

6 Conclusion

We have presented the first solution to the problem of counting the occurrences of a pattern in a text represented by an arbitrary RLCFG, which was posed by Christiansen et al. [7] in 2020 and solved only for particular cases. This required combining solutions to CFGs [44] and particular RLCFGs [7], but also new insights for the general case. The particular existing solutions required that $|B|$ is the shortest period of $\exp(A)$ in rules $A \rightarrow B^s$. While this does not hold in general RLCFGs, we proved that, except in some borderline cases that can be handled separately, the shortest periods of the pattern and of $\exp(A)$ must coincide. While the particular solutions could associate $\exp(B)$ with the period of the pattern, we must associate many strings $\exp(A)$ that share the same shortest period, and require a more sophisticated geometric data structure to collect only those that qualify for our search. Despite those complications, however, we manage to define a data structure of size $O(g_{rl})$ from a RLCFG of size g_{rl}, that counts the occurrences of $P[1..m]$ in $T[1..n]$ in time $O(m \log^{2+\epsilon} n)$ for any constant $\epsilon > 0$, the same result that existed for the simpler case of CFGs. Our approach extends the applicability of arbitrary RLCFGs to cases where only CFGs could be used, equalizing the available tools to handle both types of grammars.

Acknowledgement

We thank the reviewers for their comments, particularly one that did an exhaustive and thoughtful job to improve our presentation.

References

1. Belazzougui, D., Botelho, F.C., Dietzfelbinger, M.: Hash, displace, and compress. In: Proc. European Symposium on Algorithms (ESA). pp. 682–693. Springer (2009)
2. Bille, P., Ettienne, M.B., Gørtz, I.L., Vildhøj, H.W.: Time-space trade-offs for Lempel-Ziv compressed indexing. Theoretical Computer Science **713**, 66–77 (2018)
3. Bille, P., Landau, G.M., Raman, R., Sadakane, K., Rao, S.S., Weimann, O.: Random access to grammar-compressed strings and trees. SIAM Journal on Computing **44**(3), 513–539 (2015)
4. Bille, P., Gørtz, I.L., Sach, B., Vildhøj, H.W.: Time–space trade-offs for longest common extensions. Journal of Discrete Algorithms **25**, 42–50 (2014)
5. Charikar, M., Lehman, E., Liu, D., Panigrahy, R., Prabhakaran, M., Sahai, A., Shelat, A.: The smallest grammar problem. IEEE Transactions on Information Theory **51**(7), 2554–2576 (2005)

6 Chazelle, B.: A functional approach to data structures and its use in multidimensional searching. SIAM Journal on Computing **17**(3), 427–462 (1988)

7 Christiansen, A.R., Ettienne, M.B., Kociumaka, T., Navarro, G., Prezza, N.: Optimal-time dictionary-compressed indexes. ACM Transactions on Algorithms (TALG) **17**(1), 1–39 (2020)

8 Claude, F., Navarro, G.: Self-indexed grammar-based compression. Fundamenta Informaticae **111**(3), 313–337 (2010)

9 Claude, F., Navarro, G.: Improved grammar-based compressed indexes. In: Proc. 19th International Symposium on String Processing and Information Retrieval (SPIRE). pp. 180–192 (2012)

10 Claude, F., Navarro, G., Pacheco, A.: Grammar-compressed indexes with logarithmic search time. Journal of Computer and System Sciences **118**, 53–74 (2021)

11 Crochemore, M., Rytter, W.: Jewels of stringology: text algorithms. World Scientific (2002)

12 Ferrada, H., Gagie, T., Hirvola, T., Puglisi, S.J.: Hybrid indexes for repetitive datasets. Philosophical Transactions of the Royal Society A **372**(2016), article 20130137 (2014)

13 Ferrada, H., Kempa, D., Puglisi, S.J.: Hybrid indexing revisited. In: Proc. 20th Workshop on Algorithm Engineering and Experiments (ALENEX). pp. 1–8 (2018)

14 Fine, N.J., Wilf, H.S.: Uniqueness theorems for periodic functions. Proceedings of the American Mathematical Society **16**(1), 109–114 (1965)

15 Fredman, M.L., Komlós, J., Szemerédi, E.: Storing a sparse table with $O(1)$ worst case access time. Journal of the ACM **31**(3), 538–544 (1984)

16 Gagie, T., Gawrychowski, P., Kärkkäinen, J., Nekrich, Y., Puglisi, S.J.: A faster grammar-based self-index. In: Proc. 6th International Conference on Language and Automata Theory and Applications (LATA). pp. 240–251. LNCS 7183 (2012)

17 Gagie, T., Gawrychowski, P., Kärkkäinen, J., Nekrich, Y., Puglisi, S.J.: LZ77-based self-indexing with faster pattern matching. In: Proc. 11th Latin American Symposium on Theoretical Informatics (LATIN). pp. 731–742 (2014)

18 Gagie, T., Navarro, G., Prezza, N.: Fully-functional suffix trees and optimal text searching in BWT-runs bounded space. Journal of the ACM **67**(1), article 2 (2020)

19 Ganardi, M., Jez, A., Lohrey, M.: Balancing straight-line programs. Journal of the ACM **68**(4), 27:1–27:40 (2021)

20 Gao, Y.: Computing matching statistics on repetitive texts. In: Proc. 32nd Data Compression Conference (DCC). pp. 73–82 (2022)

21 Gawrychowski, P., Karczmarz, A., Kociumaka, T., Lacki, J., Sankowski, P.: Optimal dynamic strings. In: Proc. 29th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA). pp. 1509–1528 (2018)

22 Grossi, R., Gupta, A., Vitter, J.S.: High-order entropy-compressed text indexes. In: Proc. 14th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA). pp. 841–850 (2003)

23 Jez, A.: Approximation of grammar-based compression via recompression. Theoretical Computer Science **592**, 115–134 (2015)

24 Jez, A.: A really simple approximation of smallest grammar. Theoretical Computer Science **616**, 141–150 (2016)

25 Kärkkäinen, J., Ukkonen, E.: Lempel-Ziv parsing and sublinear-size index structures for string matching. In: Proc. 3rd South American Workshop on String Processing (WSP). pp. 141–155 (1996)

26 Karp, R.M., Rabin, M.O.: Efficient randomized pattern-matching algorithms. IBM Journal of Research and Development **2**, 249–260 (1987)

27 Kempa, D., Prezza, N.: At the roots of dictionary compression: String attractors. In: Proc. 50th Annual ACM Symposium on the Theory of Computing (STOC). pp. 827–840 (2018)

28 Kempa, D., Kociumaka, T.: Collapsing the hierarchy of compressed data structures: Suffix arrays in optimal compressed space. In: Proc. 64th IEEE Annual Symposium on Foundations of Computer Science (FOCS). pp. 1877–1886 (2023)

29 Kieffer, J.C., Yang, E.H.: Grammar-based codes: A new class of universal lossless source codes. IEEE Transactions on Information Theory **46**(3), 737–754 (2000)

30 Kociumaka, T., Navarro, G., Olivares, F.: Near-optimal search time in δ-optimal space, and vice versa. Algorithmica **86**(4), 1031–1056 (2024)

31 Kociumaka, T., Navarro, G., Prezza, N.: Toward a definitive compressibility measure for repetitive sequences. IEEE Transactions on Information Theory **69**(4), 2074–2092 (2023)

32 Kociumaka, T., Radoszewski, J., Rytter, W., Walen, T.: Internal pattern matching queries in a text and applications. In: Proc. 26th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA). pp. 532–551 (2015)

33 Kreft, S., Navarro, G.: On compressing and indexing repetitive sequences. Theoretical Computer Science **483**, 115–133 (2013)

34 Larsson, J., Moffat, A.: Off-line dictionary-based compression. Proceedings of the IEEE **88**(11), 1722–1732 (2000)

35 Lempel, A., Ziv, J.: On the complexity of finite sequences. IEEE Transactions on Information Theory **22**(1), 75–81 (1976)

36 Maruyama, S., Sakamoto, H., Takeda, M.: An online algorithm for lightweight grammar-based compression. Algorithms **5**(2), 214–235 (2012)

37 Navarro, G.: Spaces, trees and colors: The algorithmic landscape of document retrieval on sequences. ACM Computing Surveys **46**(4), article 52 (2014), 47 pages

38 Navarro, G.: Wavelet trees for all. Journal of Discrete Algorithms **25**, 2–20 (2014)

39 Navarro, G.: Indexing highly repetitive string collections, part I: Repetitiveness measures. ACM Computing Surveys **54**(2), article 29 (2021)

40 Navarro, G.: Indexing highly repetitive string collections, part II: Compressed indexes. ACM Computing Surveys **54**(2), article 26 (2021)

41 Navarro, G.: Computing MEMs on repetitive text collections. In: Proc. 34th Annual Symposium on Combinatorial Pattern Matching (CPM). p. article 22 (2023)

42 Navarro, G., Olivares, F., Urbina, C.: Balancing run-length straight-line programs. In: Proc. 29th International Symposium on String Processing and Information Retrieval (SPIRE). pp. 117–131 (2022)

43 Navarro, G., Prezza, N.: Universal compressed text indexing. Theoretical Computer Science **762**, 41–50 (2019)

44 Navarro, G.: Document listing on repetitive collections with guaranteed performance. Theoretical Computer Science **772**, 58–72 (2019)

45 Navarro, G.: Computing MEMs and relatives on repetitive text collections. ACM Transactions on Algorithms **21**(1), article 12 (2025)

46 Nevill-Manning, C., Witten, I., Maulsby, D.: Compression by induction of hierarchical grammars. In: Proc. 4th Data Compression Conference (DCC). pp. 244–253 (1994)

47 Nishimoto, T., I, T., Inenaga, S., Bannai, H., Takeda, M.: Fully dynamic data structure for LCE queries in compressed space. In: Proc. 41st International Symposium on Mathematical Foundations of Computer Science (MFCS). pp. 72:1–72:15 (2016)

48 Raskhodnikova, S., Ron, D., Rubinfeld, R., Smith, A.: Sublinear algorithms for approximating string compressibility. Algorithmica **65**, 685–709 (2013)

49 Rytter, W.: Application of Lempel-Ziv factorization to the approximation of grammar-based compression. Theoretical Computer Science **302**(1-3), 211–222 (2003)

50 Sakamoto, H.: A fully linear-time approximation algorithm for grammar-based compression. Journal of Discrete Algorithms **3**(2–4), 416–430 (2005)

51 Storer, J.A., Szymanski, T.G.: Data compression via textual substitution. Journal of the ACM **29**(4), 928–951 (1982)

52 Tsuruta, K., Köppl, D., Nakashima, Y., Inenaga, S., Bannai, H., Takeda, M.: Grammar-compressed self-index with Lyndon words. CoRR **2004.05309** (2020)