Compressed Indexing for Consecutive Occurrences

Paweł Gawrychowski ✉
Institute of Computer Science, University of Wrocław, Poland

Garance Gourdel ✉
DI/ENS, PSL Research University, IRISA Inria Rennes, France

Tatiana Starikovskaya ✉
DI/ENS, PSL Research University, Paris, France

Teresa Anna Steiner ✉
DTU Compute, Technical University of Denmark, Lyngby, Denmark

Abstract

The fundamental question considered in algorithms on strings is that of indexing, that is, preprocessing a given string for specific queries. By now we have a number of efficient solutions for this problem when the queries ask for an exact occurrence of a given pattern P. However, practical applications motivate the necessity of considering more complex queries, for example concerning near occurrences of two patterns. Recently, Bille et al. [CPM 2021] introduced a variant of such queries, called gapped consecutive occurrences, in which a query consists of two patterns P_1 and P_2 and a range $[a, b]$, and one must find all consecutive occurrences (q_1, q_2) of P_1 and P_2 such that $q_2 - q_1 \in [a, b]$. By their results, we cannot hope for a very efficient indexing structure for such queries, even if $a = 0$ is fixed (although at the same time they provided a non-trivial upper bound). Motivated by this, we focus on a text given as a straight-line program (SLP) and design an index taking space polynomial in the size of the grammar that answers such queries in time optimal up to polylog factors.

2012 ACM Subject Classification Theory of computation → Data compression; Theory of computation → Pattern matching

Keywords and phrases Compressed indexing, two patterns, consecutive occurrences

Digital Object Identifier 10.4230/LIPIcs.CPM.2023.12

Related Version *Full Version*: `https://arxiv.org/abs/2304.00887`

Funding *Paweł Gawrychowski*: Partially supported by the Bekker programme of the Polish National Agency for Academic Exchange (PPN/BEK/2020/1/00444) and the grant ANR-20-CE48-0001 from the French National Research Agency (ANR).
Garance Gourdel: Partially supported by the grant ANR-20-CE48-0001 from the French National Research Agency (ANR).
Tatiana Starikovskaya: Partially supported by the grant ANR-20-CE48-0001 from the French National Research Agency (ANR).
Teresa Anna Steiner: Supported by a research grant (VIL51463) from VILLUM FONDEN.

1 Introduction

In the indexing problem, the goal is to preprocess a string for locating occurrences of a given pattern. For a string of length N, structures such as the suffix tree [36] or the suffix array [31], use space linear in N and allow for answering such queries in time linear in the length of the pattern m. By now, we have multiple space- and time-efficient solutions for this problem (both in theory and in practice). We refer the reader to the excellent survey by Lewenstein [29] that provides an overview of some of the approaches and some of its extensions, highlighting its connection to orthogonal range searching.

However, from the point of view of possible applications, it is desirable to allow for more general queries than just locating an exact match of a given pattern in the preprocessed text, while keeping the time sublinear in the length of the preprocessed string. A very general query is locating a substring matching a regular expression. Very recently, Gibney and Thankachan [19] showed that if the Online Matrix-Vector multiplication conjecture holds, even with a polynomial preprocessing time we cannot answer regular expression query in sublinear time. A more reasonable and yet interesting query could concern occurrences of two given patterns that are closest to each other, or just close enough.

Preprocessing a string for queries concerning two patterns has been first studied in the context of document retrieval, where the goal is to preprocess a collection of strings. There, in *the two patterns document retrieval problem* the query consists of two patterns P_1 and P_2, and we must report all documents containing both of them [32]. In *the forbidden pattern query problem* we must report all documents containing P_1 but not P_2 [15]. For both problems, the asymptotically fastest linear-space solutions need as much as $\Omega(\sqrt{N})$ time to answer a query, where N is the total length of all strings [23, 22]. That is, the complexity heavily depends on the length of the strings. Larsen et al. [28] established a connection between Boolean matrix multiplication and the two problems, thus providing a conditional explanation for the high $\Omega(\sqrt{N})$ query complexity. Later, Kopelowitz et al. [27] provided an even stronger argument using a connection to the 3SUM problem. Even more relevant to this paper is the question considered by Kopelowitz and Krauthgamer [26], who asked for preprocessing a string for computing, given two patterns P_1 and P_2, their occurrences that are closest to each other. The main result of their paper is a structure constructible in $O(N^{1.5}\log^{\epsilon} N)$ time that answers such queries in $O(|P_1| + |P_2| + \sqrt{N}\log^{\epsilon} N)$, for a string of length N, for any $\epsilon > 0$. They also established a connection between Boolean matrix multiplication and this problem, highlighting a difficulty in removing the $O(\sqrt{N})$ from both the preprocessing and query time at the same time.

The focus of this paper is the recently introduced variant of the indexing problem, called *gapped indexing for consecutive occurrences*, in which a query consists of two patterns P_1 and P_2 and a range $[a, b]$, and one must find the pairs of consecutive occurrences of P_1, P_2 separated by a distance in the range $[a, b]$. Navarro and Thankanchan [33] showed that for $P_1 = P_2$ there is a $O(n \log n)$-space index with optimal query time $O(m + \text{occ})$, where $m = |P_1| = |P_2|$ and occ is the number of pairs to report, but in conclusion they noticed that extending their solution to the general case of two patterns might not be possible. Bille et al. [4] provided an evidence of hardness of the general case and established a (conditional) lower bound for gapped indexing for consecutive occurrences, by connecting its complexity to that of set intersection. This lower bound suggests that, at least for indexes of size $\tilde{O}(N)$, achieving query time better than $\tilde{O}(|P_1| + |P_2| + \sqrt{N})$ would contradict the Set Disjointness conjecture, even if $a = 0$ is fixed. In particular, obtaining query time depending mostly on the lengths of the patterns (perhaps with some additional logarithms), arguably the whole point of string indexing, is unlikely in this case.

Motivated by the (conditional) lower bound for gapped indexing for consecutive occurrences, we consider the compressed version of this problem for query intervals $[0, b]$. For exact pattern matching, there is a long line of research devoted to designing the so-called compressed indexes, that is, indexing structures with the size being a function of the length of the compressed representation of the text, see e.g. the entry in the Encyclopedia of Algorithms [30] or the Encyclopedia of Database Systems [13]. This suggests the following research direction: can we design an efficient compressed gapped index for consecutive occurrences?

The answer of course depends on the chosen compression method. With a goal to design an index that uses very little space, we focus on the most challenging setting when the compression is capable of describing a string of exponential length (in the size of its representation). An elegant formalism for such a compression method is that of straight-line programs (SLP), which are context-free grammars describing exactly one string. SLPs are known to capture the popular Lempel–Ziv compression method up to a logarithmic factor [7, 35], and at the same time provide a more convenient interface, and in particular, allow for random access in $O(\log N)$ time [5].

By now it is known that pattern matching admits efficient indexing in SLP-compressed space. Assuming a string S of length N described by an SLP with g productions, Claude and Navarro [9] designed an $O(g)$-space index for S that allows retrieving all occurrences of a pattern of length m in time $O(m^2 \log\log N + \mathrm{occ} \log g)$. Recently, several results have improved the query time bound while still using a comparable $O(g \log N)$ amount of space: Claude, Navarro and Pacheco [10] showed an index with query time $O((m^2 + \mathrm{occ}) \log g)$; Christiansen et al. [8] used strings attractors to further improve the time bound to $O(m + \mathrm{occ} \log^{\epsilon} N)$; and Díaz-Domínguez et al. [12] achieved $O((m \log m + \mathrm{occ}) \log g)$ query time.

However it is not always the case that a highly compressible string is easier to preprocess. On the negative side, Abboud et al. [1] showed that, for some problems on compressed strings, such as computing the LCS, one cannot completely avoid a high dependency on the length of the uncompressed string and that for other problems on compressed strings, such as context-free grammar parsing or RNA folding, one essentially cannot hope for anything better than just decompressing the string and working with the uncompressed representation! This is also the case for some problems related to linear algebra [2]. Hence, it was not clear to us if one can avoid a high dependency on the length of the uncompressed string in the gapped indexing for consecutive occurrences problem.

In this work, we address the lower bound of Bille et al. [4] and show that, despite the negative results by Abboud et al. [1], one can circumvent it assuming that the text is very compressible:

▶ **Theorem 1.** *For an SLP of size g representing a string S of length N, there is an $O(g^5 \log^5 N)$-space data structure that maintains the following queries: given two patterns P_1, P_2 both of length $O(m)$, and a range $[0, b]$, report all* occ *consecutive occurrences of P_1 and P_2 separated by a distance $d \in [0, b]$. The query time is $O(m \log N + (1+\mathrm{occ}) \cdot \log^4 N \log\log N)$.*

While achieving $O(g)$ space and $O(m + \mathrm{occ})$ query time would contradict the Set Disjointness conjecture by the reduction of Bille et al. [4], one might wonder if the space can be improved without increasing the query time and what is the true complexity of the problem when a is not fixed (recall that $[a, b]$ is the range limiting the distance between co-occurrences to report). While we leave improvement on space and the general case as an interesting open question, we show that in the simpler case $a = 0, b = N$ (i.e. when there is no bound on the distance between the starting positions of P_1 and P_2), our techniques do allow for $O(g^2 \log^4 N)$ space complexity, see Corollary 11[1].

Throughout the paper we assume a unit-cost RAM model of computation with word size $\Theta(\log N)$. All space complexities refer to the number of words used by a data structure.

[1] Note that the conditional lower bound of Bille et al. [4] does not hold for this simpler case.

2 Preliminaries

A *string* S of length $|S| = N$ is a sequence $S[0]S[1]\dots S[N-1]$ of characters from an alphabet Σ. We denote the *reverse* $S[N-1]S[N-2]\dots S[0]$ of S by $\mathrm{rev}(S)$. We define $S[i\dots j]$ to be equal to $S[i]\dots S[j]$ which we call a *substring* of S if $i \le j$ and to the empty string otherwise. We also use notations $S[i\dots j)$ and $S(i\dots j]$ which naturally stand for $S[i]\dots S[j-1]$ and $S[i+1]\dots S[j]$, respectively. We call a substring $S[0\dots i]$ *a prefix* of S and use a simplified notation $S[\dots i]$, and a substring $S[i\dots N-1]$ *a suffix* of S denoted by $S[i\dots]$. We say that X is a *substring* of S if $X = S[i\dots j]$ for some $0 \le i \le j \le N-1$. The index i is called an *occurrence* of X in S.

An occurrence q_1 of P_1 and an occurrence q_2 of P_2 form a *consecutive occurrence (co-occurrence)* of strings P_1, P_2 in a string S if there are no occurrences of P_1, P_2 between q_1 and q_2, formally, there should be no occurrences of P_1 in $(q_1, q_2]$ and no occurrences of P_2 in $[q_1, q_2)$. For brevity, we say that a co-occurrence is *b-close* if $q_2 - q_1 \le b$.

An integer π is a *period* of a string S of length N, if $S[i] = S[i+\pi]$ for all $i = 0, \dots, N-1-\pi$. The smallest period of a string S is called *the period* of S. We say that S is *periodic* if the period of S is at most $N/2$. We exploit the well-known corollary of the Fine and Wilf's periodicity lemma [14]:

▶ **Corollary 2.** *If there are at least three occurrences of a string Y in a string X, where $|X| \le 2|Y|$, then the occurrences of Y in X form an arithmetic progression with a difference equal to the period of Y.*

2.1 Grammars

▶ **Definition 3** (Straight-line program [25])**.** *A* straight-line program *(SLP) G is a context-free grammar (CFG) consisting of a set of non-terminals, a set of terminals, an initial symbol, and a set of productions, satisfying the following properties:*

- *A production consists of a left-hand side and a right-hand side, where the left-hand side is a non-terminal A and the right-hand side is either a sequence BC, where B, C are non-terminals, or a terminal;*
- *Every non-terminal is on the left-hand side of exactly one production;*
- *There exists a linear order $<$ on the non-terminals such that $A < B$ whenever B occurs on the right-hand side of the production associated with A.*

A *run-length straight-line program* (RLSLP) [34] additionally allows productions of form $A \to B^k$ for positive integers k, which correspond to concatenating k copies of B. If A is associated with a production $A \to a$, where a is a terminal, we denote $\mathsf{head}(A) = a$, $\mathsf{tail}(A) = \varepsilon$ (the empty string); if A is associated with a production $A \to BC$, we denote $\mathsf{head}(A) = B$, $\mathsf{tail}(A) = C$; and finally if A is associated with a production $A \to B^k$, then $\mathsf{head}(A) = B$, $\mathsf{tail}(A) = B^{k-1}$.

The *expansion* $\overline{S}$ of a sequence of terminals and non-terminals S is the string that is obtained by iteratively replacing non-terminals by the right-hand sides in the respective productions, until only terminals remain. We say that G *represents* the expansion of its initial symbol.

▶ **Definition 4** (Parse tree)**.** *The* parse tree *of a SLP (RLSLP) is a rooted tree defined as follows:*

- *The root is labeled by the initial symbol;*
- *Each internal node is labeled by a non-terminal;*
- *If S is the expansion of the initial symbol, then the ith leaf of the parse tree is labeled by a terminal $S[i]$;*
- *A node labeled with a non-terminal A that is associated with a production $A \to BC$, where B, C are non-terminals, has 2 children labeled by B and C, respectively. If A is associated with a production $A \to a$, where a is a terminal, then the node has one child labeled by a.*
- *(RLSLP only) A node labeled with non-terminal A that is associated with a production $A \to B^k$, where B is a non-terminal, has k children, each labeled by B.*

The *size* of a grammar is its number of productions. The *height* of a grammar is the height of the parse tree. We say that a non-terminal A is an *ancestor* of a non-terminal B if there are nodes u, v of the parse tree labeled with A, B respectively, and u is an ancestor of v. For a node u of the parse tree, denote by $\mathrm{off}(u)$ the number of leaves to the left of the subtree rooted at u.

▶ **Definition 5** (Relevant occurrences). *Let A be a non-terminal associated with a production $A \to \mathit{head}(A)\mathit{tail}(A)$. We say that an occurrence q of a string P in $\overline{A}$ is* relevant with a split s *if $q = |\overline{\mathit{head}(A)}| - s \le |\overline{\mathit{head}(A)}| \le q + |P| - 1$.*

For example, in Fig. 1 the occurrence $q = 3$ of $P = cab$ is a relevant occurrence in $\overline{C}$ with a split $s = 1$ but $\overline{A}$ contains no relevant occurrences of P.

▷ Claim 6. Let q be an occurrence of a string P in a string S. Consider the parse tree of an RLSLP representing S, and let w be the lowest node containing leaves $S[q], S[q+1], \dots, S[q+|P|-1]$ in its subtree, then either

1. The label A of w is associated with a production $A \to BC$, and $q - \mathrm{off}(w)$ is a relevant occurrence in $\overline{A}$; or
2. The label A of w is associated with a production $A \to B^r$ and $q - \mathrm{off}(w) = q' + r'|\overline{B}|$ for some $0 \le r' \le r$, where q' is a relevant occurrence of P in $\overline{A}$.

Proof. Assume first that A is associated with a production $A \to BC$. We then have that the subtree rooted at the left child of w (that corresponds to $\overline{B}$) does not contain $S[q+|P|-1]$ and the subtree rooted at the right child of w (that corresponds to $\overline{C}$) does not contain $S[q]$. As a consequence, $q - \mathrm{off}(w)$ is a relevant occurrence in $\overline{A}$.

Consider now the case where A is associated with a production $A \to B^r$. The leaves labeled by $S[q]$ and $S[q+|P|-1]$ belong to the subtrees rooted at different children of A. If $S[q]$ belongs to the subtree rooted at the $(r'+1)$-th child of A, then $q' = q - \mathrm{off}(w) - |\overline{B}| \cdot r'$ is a relevant occurrence of P in $\overline{A}$. ◁

▶ **Definition 7** (Splits). *Consider a non-terminal A of an RLSLP G. If it is associated with a production $A \to BC$, define*

$$\mathrm{Splits}(A, P) = \mathrm{Splits}_{\mathrm{rev}}(A, P) = \{s : q \textit{ is a relevant occurrence of } P \textit{ in } \overline{A} \textit{ with a split } s\}.$$

If A is associated with a rule $A \to B^k$, define

$$\mathrm{Splits}(A, P) = \{s : q \textit{ is a relevant occurrence of } P \textit{ in } \overline{A} \textit{ with a split } s\};$$
$$\mathrm{Splits}_{\mathrm{rev}}(A, P) = \{|P| - s : q \textit{ is a relevant occurrence of } \mathrm{rev}(P) \textit{ in } \mathrm{rev}(\overline{A}) \textit{ with split } s\}.$$

Define $\mathrm{Splits}(G, P)$ $(\mathrm{Splits}_{\mathrm{rev}}(G, P))$ *to be the union of* $\mathrm{Splits}(A, P)$ $(\mathrm{Splits}_{\mathrm{rev}}(A, P))$ *over all non-terminals A in G, and* $\mathrm{Splits}'(G, P) = \mathrm{Splits}(G, P) \cup \mathrm{Splits}_{\mathrm{rev}}(G, P)$.

We need the following lemma, which can be derived from Gawrychowski et al. [18]:

▶ **Lemma 8.** *Let G be an SLP of size g representing a string S of length N, where $g \leq N$. There exists a Las Vegas algorithm that builds a RLSLP G' of size $g' = O(g \log N)$ of height $h = O(\log N)$ representing S in time $O(g \log N)$ with high probability. This RLSLP has the following additional property: For a pattern P of length m, we can in $O(m \log N)$ time provide a certificate that P does not occur in S, or compute the set $\text{Splits}'(G', P)$. In the latter case, $|\text{Splits}'(G', P)| = O(\log N)$.*

2.2 Compact Tries

We assume the reader to be familiar with the definition of a compact trie (see e.g. [21]). Informally, a trie is a tree that represents a lexicographically ordered set of strings. The edges of a trie are labeled with strings. We define the label $\lambda(u)$ of a node u to be the concatenation of labels on the path from the root to u and an interval $I(u)$ to be the interval of the set of strings starting with $\lambda(u)$. From the implementation point of view, we assume that a node u is specified by the interval $I(u)$. The *locus* of a string P is the minimum depth node u such that P is a prefix of $\lambda(u)$.

The standard tree-based implementation of a trie for a generic set of strings $\mathcal{S} = \{S_1, \ldots, S_k\}$ takes $\Theta\left(\sum_{i=1}^{k} |S_i|\right)$ space. Given a pattern P of length m and $\tau > 0$ suffixes $Q_1, \ldots, Q_\tau$ of P, the trie allows retrieving the ranges of strings in (the lexicographically-sorted) $\mathcal{S}$ prefixed by $Q_1, \ldots, Q_\tau$ in $O(m^2)$ time. However, in this work, we build the tries for very special sets of strings only, which allows for a much more efficient implementation based on the techniques of Christiansen et al. [8], the proof is given in Appendix A:

▶ **Lemma 9.** *Given an RLSLP G of size g and height h. Assume that every string in a set $\mathcal{S}$ is either a prefix or a suffix of the expansion of a non-terminal of G or its reverse. The trie for $\mathcal{S}$ can be implemented in space $O(|\mathcal{S}|)$ to maintain the following queries in $O(m + \tau \cdot (h + \log m))$ time: Given a pattern P of length m and suffixes Q_i of P, $1 \leq i \leq \tau$, find, for each i, the interval of strings in the (lexicographically sorted) $\mathcal{S}$ prefixed by Q_i.*

3 Relevant, extremal, and predecessor occurrences in a non-terminal

In this section, we present a data structure that allows various efficient queries, which we will need to prove Theorem 1. We also show how it can be leveraged for an index in the simpler case of consecutive occurrences ($a = 0, b = N$). Recall that the text S is a string of length N represented by an SLP G of size g. By applying Lemma 8, we transform G into an RLSLP G' of size $g' = O(g \log N)$ and depth $h = O(\log N)$ representing S, which we fix from now on. We start by showing that G' can be processed in small space to allow multiple efficient queries:

▶ **Theorem 10.** *There is a $O(g^2 \log^4 N)$-space data structure for G' that given a pattern P of length m can preprocess it in $O(m \log N + \log^2 N)$ time to support the following queries for a given non-terminal A of G':*

1. *Report the sorted set of relevant occurrences of P in $\overline{A}$ in $O(\log N)$ time;*
2. *Decide whether there is an occurrence of P in $\overline{A}$ in $O(\log N \log \log N)$ time;*
3. *Report the leftmost and the rightmost occurrences of P in $\overline{A}$, $\overline{\mathsf{head}(A)}$, and $\overline{\mathsf{tail}(A)}$ in $O(\log^2 N \log \log N)$ time;*
4. *Given a position p, find the rightmost (leftmost) occurrence $q \leq p$ ($q \geq p$) of P in $\overline{A}$ in $O(\log^3 N \log \log N)$ time (predecessor/successor).*

Before we proceed to the proof, let us derive a data structure to report all consecutive occurrences (co-occurrences) of a given pair of patterns.

▶ **Corollary 11.** *For an SLP of size g representing a string S of length N, there is an $O(g^2 \log^4 N)$-space data structure that supports the following queries: given two patterns P_1, P_2 both of length $O(m)$, report all* occ *co-occurrences of P_1 and P_2 in S. The query time is $O(m \log N + (1 + \text{occ}) \cdot \log^3 N \log\log N)$.*

Proof. We exploit the data structure of Theorem 10 for G'. To report all co-occurrences of P_1, P_2 in S, we preprocess P_1, P_2 in $O(m \log N + \log^2 N)$ time and then proceed as follows. Suppose that we want to find the leftmost co-occurrence of P_1 and P_2 in the string $S[i \ldots]$, where at the beginning $i = 0$. We find the leftmost occurrence q_1' of P_1 with $q_1' \geq i$ (if it exists) by a successor query on the initial symbol of G' (the expansion of which is the entire string S). Then we find the leftmost occurrence q_2 of P_2 with $q_2 \geq q_1'$ (if it exists) by a successor query and the rightmost occurrence q_1 of P_1 with $q_1 \leq q_2$ by a predecessor query. If either q_1' or q_2 do not exist, then there are no more co-occurrences in $S[i \ldots]$. Otherwise, clearly, (q_1, q_2) is a co-occurrence, and there can be no other co-occurrences starting in $S[i \ldots q_2]$. In this case, we return (q_1, q_2) and set $i = q_2 + 1$. The running time of the retrieval phase is $O(\log^3 N \log\log N \cdot (\text{occ} + 1))$, since we use at most three successor/predecessor queries to either output a new co-occurrence or decide that there are no more co-occurrences. ◀

3.1 Proof of Theorem 10

The data structure consists of two compact tries T_{pre} and T_{suf} defined as follows. For each non-terminal A, we store $\text{rev}(\overline{\mathsf{head}(A)})$ in T_{pre} and $\overline{\mathsf{tail}(A)}$ in T_{suf}. We augment T_{pre} and T_{suf} by computing their heavy path decomposition:

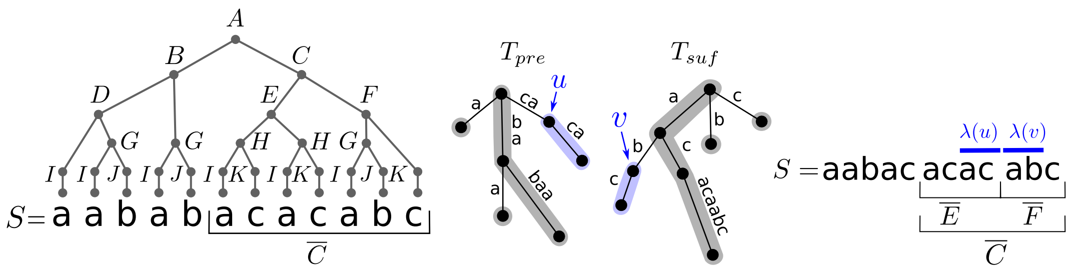
▶ **Definition 12.** *The* heavy path *of a trie T is the path that starts at the root of T and at each node v on the path branches to the child with the largest number of leaves in its subtree (*heavy *child), with ties broken arbitrarily. The heavy path decomposition is a set of disjoint paths defined recursively, namely it is defined to be a union of the singleton set containing the heavy path of T and the heavy path decompositions of the subtrees of T that hang off the heavy path.*

For each non-terminal A of G', a heavy path h_{pre} in T_{pre}, and a heavy path h_{suf} in T_{suf}, we construct a multiset of points $\mathcal{P}(A, h_{pre}, h_{suf})$. For every non-terminal A' and nodes $u \in h_{pre}$, $v \in h_{suf}$ the multiset contains a point $(|\lambda(u)|, |\lambda(v)|)$ iff A', u, v satisfy the following properties:

1. A is an ancestor of A';
2. $I(u)$ contains $\text{rev}(\overline{\mathsf{head}(A')})$ and $I(v)$ contains $\overline{\mathsf{tail}(A')}$.
3. u, v are the lowest nodes in h_{pre}, h_{suf}, respectively, satisfying Property 2.

(See Fig. 1.) The set $P(A, h_{pre}, h_{suf})$ is stored in a two-sided 2D orthogonal range emptiness data structure [29, 6] which occupies $O(|\mathcal{P}(A, h_{pre}, h_{suf})|)$ space. Given a 2D range of the form $[\alpha, \infty] \times [\beta, \infty]$, it allows to decide whether the range contains a point in $\mathcal{P}(A, h_{pre}, h_{suf})$ in $O(\log\log N)$ time.

▷ Claim 13. The data structure occupies $O(g^2 \log^4 N)$ space.



(a) Parse tree of G'. **(b)** Searching for cab with a split $s=1$.

Figure 1 A string $S=$ aababacacabc is generated by an SLP G'. Nodes u and v are the loci of c and ab in T_{pre} and T_{suf} respectively. The heavy paths h_{pre} in T_{pre} and h_{suf} in T_{suf} are shown in blue. We have $(2,2) \in \mathcal{P}(A, h_{pre}, h_{suf})$ corresponding to C, u, v.

Proof. Each non-terminal A' has at most g' distinct ancestors and each root-to-leaf path in T_{pre} or T_{suf} crosses $O(\log g')$ heavy paths (as each time we switch heavy paths, the number of leaves in the subtree of the current node decreases by at least a factor of two). As a corollary, each non-terminal creates $O(g' \log^2 g') = O(g \log^3 N)$ points across all orthogonal range emptiness data structures. ◁

When we receive a pattern P, we compute $\text{Splits}'(G',P)$ via Lemma 8 in $O(m \log N)$ time or provide a certificate that P does not occur in S, in which case there are no occurrences of P in the expansions of the non-terminals of G'. Recall that $|\text{Splits}'(G',P)| \in O(\log N)$. We then sort $\text{Splits}'(G',P)$ in $O(\log^2 N)$ time (a technicality which will allow us reporting relevant occurrences sorted without time overhead). Finally, we compute, for each $s \in \text{Splits}'(G',P)$, the interval of strings in T_{pre} prefixed by $\text{rev}(P[\ldots s])$ (which is the interval $I(u)$ for the locus u of $\text{rev}(P[\ldots s])$ in T_{pre}) and the interval of strings in T_{suf} prefixed by $P(s\ldots]$ (which is the interval $I(u)$ for the locus u of $P(s\ldots]$ in T_{suf}). By Lemma 9, with $\tau = |\text{Splits}'(G',P)| = O(\log N)$ and $h = O(\log N)$, this step takes $O(m + \log^2 N)$ time.

Reporting relevant occurrences is easy: by definition, each relevant occurrence q of P in $\overline{A}$ is equal to $|\overline{\mathsf{head}(A)}| - s$ for some $s \in \text{Splits}'(G',P)$ such that $\text{rev}(P[\ldots s])$ is a prefix of $\text{rev}(\overline{\mathsf{head}(A)})$ and $P(s\ldots]$ is a prefix of $\overline{\mathsf{tail}(A)}$. As we already know the intervals of the strings in T_{suf} and T_{pre} starting with $\text{rev}(P[\ldots s])$ and $P(s\ldots]$, respectively, both conditions can be checked in constant time per split, or in $O(|\text{Splits}'(G',P)|) = O(\log N)$ time overall. Note that since $\text{Splits}'(G',P)$ are sorted, the relevant occurrences are reported sorted as well.

We now explain how to answer emptiness queries on a non-terminal:

▷ Claim 14. Let A be a non-terminal labeling a node in the parse tree of G'. We can decide whether $\overline{A}$ contains an occurrence of P in $O(\log N \log\log N)$ time.

Proof. Below we show that P occurs in $\overline{A}$ iff there exists a split $s \in \text{Splits}'(G',P)$ such that for u being the locus of $\text{rev}(P[\ldots s])$ in T_{pre} and v the locus of $P(s\ldots]$ in T_{suf}, for h_{pre} the heavy path containing u in $T_{pre}$and h_{suf} the heavy path containing v in T_{suf}, the rectangle $[|\lambda(u)|, +\infty] \times [|\lambda(v)|, +\infty]$ contains a point from $\mathcal{P}(A, h_{pre}, h_{suf})$. Before we proceed to the proof, observe that by the bound on $|\text{Splits}'(G',P)|$ this allows us to decide whether P occurs in $\overline{A}$ in $O(\log N)$ range emptiness queries, which results in $O(\log N \log\log N)$ query time.

Assume that $[|\lambda(u)|, +\infty] \times [|\lambda(v)|, +\infty]$ contains a point $(x,y) \in \mathcal{P}(A, h_{pre}, h_{suf})$ corresponding to a non-terminal A'. By construction, A is an ancestor of A', the subtree of u contains a leaf corresponding to $\text{rev}(\overline{\mathsf{head}(A')})$ and the subtree of v contains a leaf corresponding to $\overline{\mathsf{tail}(A')}$. Consequently, $\overline{A'}$ contains an occurrence of P, which implies that $\overline{A}$ contains an occurrence of P. To show the reverse direction, let $\ell = \mathrm{off}(u) + 1$ and $r = \mathrm{off}(u) + |\overline{A}|$, i.e. $S[\ell \dots r] = \overline{A}$. The string $\overline{A}$ contains an occurrence $\overline{A}[q \dots q + |P|)$ of P iff $S[\ell + q \dots \ell + q + |P|)$ is an occurrence of P in S. From Claim 6 it follows that if w is the lowest node in the parse tree of G' that contains leaves $S[\ell + q], \dots, S[\ell + q + |P| - 1]$ in its subtree and A' is its label, then there exists a split $s \in \mathrm{Splits}'(G', P)$ such that $\mathrm{rev}(P[\dots s])$ is a prefix of $\mathrm{rev}(\overline{\mathsf{head}(A')})$ and $P(s \dots]$ of $\overline{\mathsf{tail}(A')}$. By definition of u and v, the leaf of T_{pre} labeled with $\mathrm{rev}(\overline{\mathsf{head}(A')})$ belongs to $I(u)$ and the leaf of T_{suf} labeled with $\overline{\mathsf{tail}(A')}$ belongs to $I(v)$. Let h_{pre} (h_{suf}) be the heavy path in $T_{pre}(T_{suf})$ containing u (v) and (x, y) be the point in $\mathcal{P}(A, h_{pre}, h_{suf})$ created for A'. As $|\lambda(u)| \le x$ and $|\lambda(v)| \le y$, the rectangle $[|\lambda(u)|, +\infty] \times [|\lambda(v)|, +\infty]$ is not empty. ◁

It remains to explain how to retrieve the leftmost/rightmost occurrences in a non-terminal, as well as to answer predecessor/successor queries. The main idea for all four types of queries is to start at any node of the parse tree of G' labeled by A and recurse down via emptiness queries and case inspection. Since the length of the expansion decreases each time we recurse from a non-terminal to its child and the height of G' is $h = O(\log N)$, this allows to achieve the desired query time. We provide full details in Appendix B.

4 Compressed Indexing for Close Co-occurrences

In this section, we show our main result, Theorem 1. Recall that S is a string of length N represented by an SLP G of size g. We start by applying Lemma 8 to transform G into an RLSLP G' of size $g' = O(g \log N)$ and height $h = O(\log N)$ representing S.

The query algorithm uses the following strategy: first, it identifies all non-terminals of G' such that their expansion contains a b-close relevant co-occurrence, where a relevant co-occurrence is defined similarly to a relevant occurrence:

▶ **Definition 15** (Relevant co-occurrence). *Let A be a non-terminal of G'. We say that a co-occurrence (q_1, q_2) of P_1, P_2 in $\overline{A}$ is* relevant *if $q_1 \le |\overline{\mathsf{head}(A)}| \le q_2 + |P_2| - 1$.*

Second, it retrieves all b-close relevant co-occurrences in each of those non-terminals, and finally, reports all b-close co-occurrences by traversing the (pruned) parse tree of G', which is possible due to the following claim:

▷ Claim 16. Assume that P_2 is not a substring of P_1, and let (q_1, q_2) be a co-occurrence of P_1, P_2 in a string S. In the parse tree of G', there exists a unique node u such that either

1. Its label A is associated with a production $A \to BC$, and $(q_1 - \mathrm{off}(u), q_2 - \mathrm{off}(u))$ is a relevant co-occurrence of P_1, P_2 in $\overline{A}$;
2. Its label A is associated with a production $A \to B^k$, $q_1 - \mathrm{off}(u) = q_1' + k'|\overline{B}|$, $q_2 - \mathrm{off}(u) = q_2' + k'|\overline{B}|$ for some $0 \le k' \le k$, where (q_1', q_2') is a relevant co-occurrence of P_1, P_2 in $\overline{A}$.

Proof. Let A be the label of the lowest node u in the parse tree that contains leaves $S[q_1], S[q_1 + 1], \dots, S[q_2 + |P_2| - 1]$ in its subtree. Because P_2 is not a substring of P_1, A cannot be associated with a production $A \to a$. By definition, $S[\mathrm{off}(u) + 1]$ is the leftmost leaf in the subtree of this node.

Assume first that A is associated with a production $A \to BC$. We then have that the subtree rooted at the left child of u (labeled by B) does not contain $S[q_2 + |P_2| - 1]$ and the subtree rooted at the right child of u (labeled by C) does not contain $S[q_1]$. As a consequence, $(q_1 - \mathrm{off}(u), q_2 - \mathrm{off}(u))$ is a relevant co-occurrence of P_1, P_2 in $\overline{A}$.

Consider now the case where A is associated with a production $A \to B^k$. The leaves labeled by $S[q_1]$ and $S[q_2 + |P_2| - 1]$ belong to the subtrees rooted at different children of A. If $S[q_1]$ belongs to the subtree rooted at the k'-th child of A, then $(q_1 - \text{off}(u) - |\overline{B}| \cdot (k' - 1), q_2 - \text{off}(u) - |\overline{B}| \cdot (k' - 1))$ is a relevant co-occurrence of P_1, P_2 in $\overline{A}$. ◁

4.1 Combinatorial observations

Informally, we define a set of $O(g^2)$ strings and show that for any patterns P_1, P_2 there are two strings S_1, S_2 in the set with the following property: whenever the expansion of a non-terminal A in G' contains a pair of occurrences P_1, P_2 forming a relevant co-occurrence, there are occurrences of S_1, S_2 in the proximity. This will allow us to preprocess the non-terminals of G' for occurrences of the strings in the set and use them to detect b-close relevant co-occurrences of P_1, P_2.

Consider two tries, T_{pre} and T_{suf}: For each production of G' of the form $A \to BC$, we store $\overline{C}$ in T_{suf} and $\text{rev}(\overline{B})$ in T_{pre}. For each production of the form $A \to B^k$, we store $\overline{B}$, $\overline{B^2}$, $\overline{B^{k-2}}$, and $\overline{B^{k-1}}$ in T_{suf} and the reverses of those strings in T_{pre}. For $j \in \{1, 2\}$ and $s \in \text{Splits}'(G', P_j)$ define $S_j(s) = \text{rev}(U)V$, where U is the label of the locus of $\text{rev}(P_j[\ldots s])$ in T_{pre} and V is the label of the locus of $P_j(s \ldots]$ in T_{suf}. Let $l_j(s) = |\text{rev}(U)|$ and $\Delta_j(s) = l_j(s) - s$.

Consider a non-terminal A such that its expansion $\overline{A}$ contains a relevant co-occurrence (q_1, q_2) of P_1, P_2.

▷ Claim 17. There exists $s \in \text{Splits}'(G', P_2)$ such that $p_2 = q_2 - \Delta_2(s)$ is an occurrence of $S_2(s)$ in $\overline{A}$ and $[p_2, p_2 + |S_2(s)|) \supseteq [q_2, q_2 + |P_2|)$.

Proof. Below we show that there exists a descendant A' of A and a split $s \in \text{Splits}'(G', P_2)$ such that either $\text{rev}(P_2[\ldots s])$ is a prefix of $\text{rev}(\overline{\mathsf{head}(A')})$ and $P_2(s \ldots]$ is a prefix of $\overline{\mathsf{tail}(A')}$, or A' is associated with a rule $A' \to (B')^k$, $\text{rev}(P_2[\ldots s])$ is a prefix of $\text{rev}(\overline{(B')^2})$ and $P_2(s \ldots]$ is a prefix of $\overline{(B')^{k-2}}$. The claim follows by the definition of T_{pre}, T_{suf}, and $S_2(s)$.

If q_2 is relevant in $\overline{A}$, there exists a split $s \in \text{Splits}'(G', P_2)$ such that $\text{rev}(P_2[\ldots s])$ is a prefix of $\text{rev}(\overline{\mathsf{head}(A)})$ and $P_2(s \ldots]$ is a prefix of $\overline{\mathsf{tail}(A)}$ by definition. If q_2 is not relevant, then $q_2 \geq |\overline{\mathsf{head}(A)}|$ by the definition of a co-occurrence. By Claim 6, there is a descendant A' of A corresponding to a substring $\overline{A}[\ell \ldots r]$ for which either $(q_2 - \ell)$ is relevant (and then we can repeat the argument above), or A' is associated with a rule $A' \to (B')^k$ and $(q_2 - \ell) - k' \cdot |\overline{B'}|$ is relevant, for some $0 \leq k' \leq k$. Consider the latter case. If $A' = A$, then $k' = 1$, as otherwise $q_1 < q_2' = q_2 - |\overline{B'}| < q_2$ is an occurrence of P_2 in $\overline{A}$ contradicting the definition of a co-occurrence (recall that (q_1, q_2) is a relevant co-occurrence and hence by definition $q_1 < |\overline{\mathsf{head}(A)}|$), and therefore $s = |\overline{(B')^2}| - q_2 + \ell \in \text{Splits}'(G', P_2)$, $\text{rev}(P_2[\ldots s])$ is a prefix of $\text{rev}(\overline{(B')^2})$ and $P_2(s \ldots]$ is a prefix of $\overline{(B')^{k-2}}$. If $A' \neq A$, then we can analogously conclude that $k' = 0$, which implies $s = |\overline{B'}| - q_2 + \ell \in \text{Splits}'(G', P_2)$, $\text{rev}(P_2[\ldots s])$ is a prefix of $\text{rev}(\overline{B'})$ and $P_2(s \ldots]$ is a prefix of $\overline{(B')^{k-1}}$. ◁

As the definition of a co-occurrence is not symmetric, q_1 does not enjoy the same property. However, a similar claim can be shown:

▶ **Lemma 18.** *There exists $s \in \text{Splits}'(G', P_1)$ and an occurrence p_1 of $S_1(s)$ in $\overline{A}$ such that $[p_1, p_1 + |S_1(s)|) \supseteq [q_1, q_1 + |P_1|)$ and at least one of the following holds:*

1. *$q_1 - \Delta_1(s)$ is an occurrence of $S_1(s)$;*
2. *q_2 is a relevant occurrence of P_2 in $\overline{A}$, the period of $S_1(s)$ equals the period π_1 of P_1, and there exists an integer k such that $p_1 = q_1 - \Delta_1(s) - \pi_1 \cdot k$ and $q_2 + \pi_1 - 1 \leq p_1 + |S_1(s)| - 1 \leq q_2 + |P_2| - 1$.*

Proof. If q_1 is a relevant occurrence of P_1 in A with a split $s \in \mathrm{Splits}'(G', P_1)$, then $\mathrm{rev}(P_1[\ldots s])$ is a prefix of $\mathrm{rev}(\overline{\mathsf{head}(A)})$ and $P_1(s\ldots]$ is a prefix of $\overline{\mathsf{tail}(A)}$ and therefore the first case holds by the definition of T_{pre} and T_{suf}.

Otherwise, by Claim 6, there is a descendant A' of $\mathsf{head}(A)$ corresponding to a substring $\overline{A}[\ell\ldots r]$ for which either $(q_1 - \ell)$ is relevant (and then we can repeat the argument above), or A' is associated with a rule $A' \to (B')^k$ and $(q_1 - \ell) - k' \cdot |\overline{B'}|$, for some $0 \le k' \le k$, is a relevant occurrence of P_1 in $\overline{A'}$ with a split $s \in \mathrm{Splits}'(G', P_1)$. Consider the latter case. We must have (1) $q_1 + |P_1| - 1 + |\overline{B'}| \ge r$ or (2) $q_1 + |\overline{B'}| - 1 \ge q_2$, because if both inequalities do not hold, then $q_1 < q_1 + |\overline{B'}| \le q_2$ is an occurrence of P_1 in $\overline{A}$, which contradicts the definition of a co-occurrence. Additionally, if (1) holds, then by definition there exists a split $s' \in \mathrm{Splits}'(G', P_1)$ (which might be different from the split s above) such that $\mathrm{rev}(P_1[\ldots s'])$ is a prefix of $\mathrm{rev}(\overline{(B')^{r-1}})$ and $P_1(s'\ldots]$ is a prefix of $\overline{B'}$ and we fall into the first case of the lemma.

From now on, assume that (2) holds and (1) does not. Since $q_1 + |\overline{B'}| \le r \le |\overline{\mathsf{head}(A)}|$ and (q_1, q_2) is a relevant co-occurrence, q_2 must be a relevant occurrence of P_2 in $\overline{A}$. If $|P_1| - s \le |\overline{(B')^2}|$, then $\mathrm{rev}(P_1[\ldots s])$ is a prefix of $\mathrm{rev}(\overline{B'})$ and $P_1(s\ldots]$ is a prefix of $\overline{(B')^2}$ and therefore $q_1 - \Delta_1(s)$ is an occurrence of $S_1(s)$. Otherwise, by Fine and Wilf's periodicity lemma [14], the periods of $\overline{A'}$, P_1, and $S_1(s)$ are equal, since P_1 and hence $S_1(s)$ span at least two periods of $\overline{A'}$. By periodicity, $S_1(s)$ occurs at positions $q_1 - \Delta_1(s) - |\overline{B'}| \cdot k$ of $\overline{A}$. Let p_1 be the leftmost of these positions which satisfies $p_1 + |S_1(s)| - 1 \ge q_1 + |P_1| - 1$. This position is well-defined as (1) does not hold, and furthermore $[q_1, q_1 + |P_1|) \subseteq [p_1, p_1 + |S_1(s)|)$ as $s \le l_1(s)$ and $|S_1(s)| - l_1(s) \ge |P_1| - s$. We have $p_1 = q_1 - \Delta_1(s) - \pi_1 \cdot k''$ for some integer k'' (as $|\overline{B'}|$ is a multiple of π_1), and $q_2 + \pi_1 - 1 \le q_1 + 2|\overline{B'}| - 1 \le q_1 + |P_1| - 1 \le p_1 + |S_1(s)| - 1 \le r < q_2 + |P_2| - 1$, where the last inequality holds as q_2 is a relevant occurrence in $\overline{A}$. The claim of the lemma follows. ◀

We summarize Claim 17 and Lemma 18:

▶ **Corollary 19.** *Let (q_1, q_2) be a co-occurrence of P_1, P_2 in the expansion of a non-terminal A. There exist splits $s_1 \in \mathrm{Splits}'(G', P_1), s_2 \in \mathrm{Splits}'(G', P_2)$ and occurrences p_1 of $S_1(s_1)$ and p_2 of $S_2(s)$, where $[p_1, p_1 + |S_1(s_1)|) \supseteq [q_1, q_1 + |P_1|)$ and $[p_2, p_2 + |S_2(s_2)|) \supseteq [q_2, q_2 + |P_2|)$, such that at least one of the following holds:*

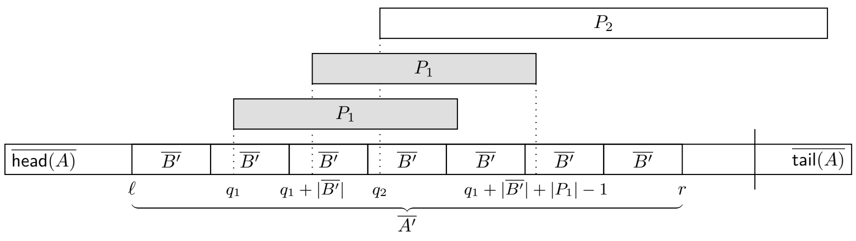
1. *The occurrence p_1 is either relevant or $p_1 + |S_1(s_1)| - 1 \le |\overline{\mathsf{head}(A)}|$. The occurrence p_2 is either relevant or $p_2 > |\overline{\mathsf{head}(A)}|$. Additionally, $p_1 = q_1 - \Delta_1(s_1)$ and $p_2 = q_2 - \Delta_2(s_2)$.*
2. *The occurrence p_2 is relevant and $p_1 \le |\overline{\mathsf{head}(A)}|$. Additionally, $p_2 = q_2 - \Delta_2(s_2)$, the period of $S_1(s)$ equals the period π_1 of P_1, and there exists an integer k such that $p_1 = q_1 - \Delta_1(s_1) - \pi_1 \cdot k$ and $p_2 + \pi_1 - 1 \le p_1 + |S_1(s_1)| - 1 \le p_2 + |S_2(s_2)| - 1$.*

The reverse observation holds as well:

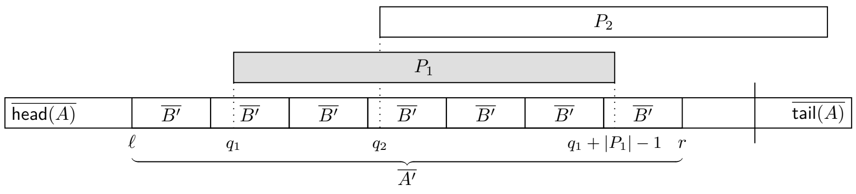
▶ **Observation 20.** *If p_j is an occurrence of $S_j(s)$ in $\overline{A}$, $j = 1, 2$, then $q_j = p_j + \Delta_j(s)$ is an occurrence of P_j. Furthermore, if $S_1(s)$ is periodic with period π_1, then $q_1 + \pi_1 \cdot k$, $0 \le k \le \lfloor (|S_1(s)| - q_1 - |P_1|)/\pi_1 \rfloor$, are occurrences of P_1 in $\overline{A}$.*

Finally, the following trivial observation will be important for upper bounding the time complexity of our query algorithm:

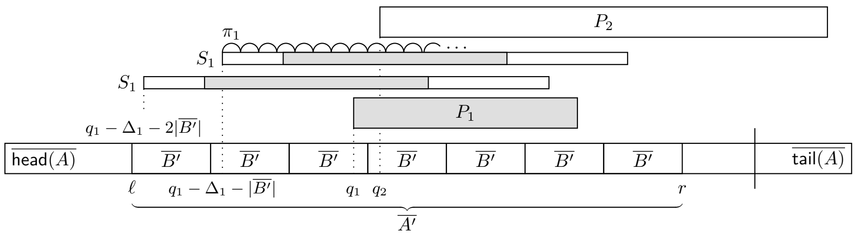
▶ **Observation 21.** *If a string contains a pair of occurrences (q_1, q_2) of P_1 and P_2 such that $0 \le q_2 - q_1 \le b$, then it contains a b-close co-occurrence of P_1 and P_2.*


P_2
P_1
P_1
$\mathsf{head}(\overline{A})$
$\overline{B'}$
$\mathsf{tail}(\overline{A})$
ℓ
q_1
$q_1 + |\overline{B'}|$
q_2
$q_1 + |\overline{B'}| + |P_1| - 1$
r
$\overline{A'}$

(a) If neither (1) nor (2), then (q_1, q_2) is not consecutive.



(b) (1) holds and (2) does not.


P_2
π_1
S_1
S_1
$q_1 - \Delta_1 - 2|\overline{B'}|$
P_1
$\mathsf{head}(\overline{A})$
$\overline{B'}$
$\mathsf{tail}(\overline{A})$
ℓ
$q_1 - \Delta_1 - |\overline{B'}|$
q_1
q_2
r
$\overline{A'}$

(c) (2) holds, (1) does not, and $|P_1| - s \geq \overline{(B')^2}$.

Figure 2 Subcases of Lemma 18.

4.2 Index

The first part of the index is the data structure of Theorem 10 and the index of Christiansen et al. [8]:

▶ **Fact 22** ([8, Introduction and Theorem 6.12]). *There is a $O(g \log^2 N)$-space data structure that can find the* occ *occurrences of any pattern $P[1 \dots m]$ in S in time $O(m + \text{occ})$.*

The second part of the index are the tries T_{pre} and T_{suf}, augmented as explained below. Consider a quadruple (u_1, u_2, v_1, v_2), where u_1 and u_2 are nodes of T_{pre} and v_1 and v_2 are nodes of T_{suf}. Let U_1, U_2, V_1, V_2 be the labels of u_1, u_2, v_1, v_2, respectively. Define $S_1 = \text{rev}(U_1)V_1$ and $S_2 = \text{rev}(U_2)V_2$, and let $l_1 = |\text{rev}(U_1)|$ and $l_2 = |\text{rev}(U_2)|$.

First, we store a binary search tree $\mathcal{T}_1(u_1, u_2, v_1, v_2)$ that for each non-terminal A contains at most six integers $d = p_2 - p_1$, where p_1, p_2 are occurrences of S_1, S_2 in $\overline{A}$, satisfying at least one of the below:

1. p_1 is the rightmost occurrence of S_1 such that $p_1 + |S_1| - 1 < |\overline{\mathsf{head}(A)}|$ and p_2 is the leftmost occurrence of S_2 such that $p_2 \geq |\overline{\mathsf{head}(A)}|$;
2. p_1 is a relevant occurrence of S_1 with a split l_1 and p_2 is the leftmost occurrence of S_2 such that $p_2 \geq |\overline{\mathsf{head}(A)}|$;
3. p_1 is a relevant occurrence of S_1 with a split l_1, p_2 is a relevant occurrence of S_2 with a split l_2;
4. p_2 is a relevant occurrence of S_2 with a split l_2 and p_1 is the rightmost occurrence of S_1 such that $p_1 + |S_1| - 1 < p_2$;
5. p_2 is a relevant occurrence of S_2 with a split l_2 and p_1 is the leftmost or second leftmost occurrence of S_1 in $\overline{\mathsf{head}(A)}$ such that $p_1 < p_2 \leq p_1 + |S_1| - 1 < p_2 + |S_2| - 1$.

Second, we store a list of non-terminals $\mathcal{L}(u_2, v_2)$ such that their expansion contains a relevant occurrence of S_2 with a split l_2. Additionally, for every $k \in [0, \log N]$, we store, if defined:

1. The rightmost occurrence p_1 of S_1 in S_2 such that $p_1 + (|S_1| - 1) \leq l_2 - 2^k$;
2. The leftmost occurrence p_1' of S_1 in S_2 such that $p_1' \leq l_2 - 2^k \leq p_1' + |S_1| - 1$;
3. The rightmost occurrence p_1'' of S_1 in S_2 such that $p_1'' \leq l_2 - 2^k \leq p_1'' + |S_1| - 1$.

Finally, we compute and memorize the period π_1 of S_1. If the period is well-defined (i.e., S_1 is periodic), we build a binary search tree $\mathcal{T}_2(u_1, u_2, v_1, v_2)$. Consider a non-terminal A containing a relevant occurrence p_2 of S_2 with a split l_2. Let p_1 be the leftmost occurrence of S_1 such that $p_1 \leq p_2 \leq p_1 + |S_1| - 1 \leq p_2 + |S_2| - 1$ and p_1' the rightmost. If p_1 and p_1' exist (p_1 might be equal to p_1') and $p_1' + |S_1| - 1 \geq p_2 + \pi_1 - 1$, we add an integer $(p_1' - p_1)/\pi_1$ to the tree and associate it with A. We also memorize a number $\mathrm{ov}(S_1, S_2) = p_2 - p_1'$, which does not depend on A by Corollary 2 and therefore is well-defined (it corresponds to the longest prefix of S_2 periodic with period π_1).

▷ Claim 23. The data structure occupies $O(g^5 \log^5 N)$ space.

Proof. The data structure of Theorem 10 occupies $O(g^2 \log^4 N)$ space. The index of Christiansen et al. occupies $O(g \log^2 N)$ space. The tries, by Lemma 9, use $O(g') = O(g \log N)$ space. There are $O((g')^4)$ quadruples (u_1, u_2, v_1, v_2) and for each of them the trees take $O(g')$ space. The arrays of occurrences of S_1 in S_2 use $O(\log N)$ space. Therefore, overall the data structure uses $O(g^5 \log^5 N)$ space. ◁

4.3 Query

Recall that a query consists of two strings P_1, P_2 of length at most m each and an integer b, and we must find all b-close co-occurrences of P_1, P_2 in S, let occ be their number.

We start by checking whether P_2 occurs in P_1 using a linear-time and constant-space pattern matching algorithm such as [11]. If it is, let q_2 be the position of the first occurrence. If $q_2 > b$, then there are no b-close co-occurrences of P_1, P_2 in S. Otherwise, to find all b-close co-occurrences of P_1, P_2 in S (that *always* consist of an occurrence of P_1 in S and the first occurrence of P_2 in P_1), it suffices to find all occurrences of P_1 in S, which we do using the index of Christiansen et al. [8] in time $O(|P_1| + \mathrm{occ}) = O(m + \mathrm{occ})$.

From now on, assume that P_2 is not a substring of P_1. Let $\mathcal{N}$ be the set of all non-terminals in G' such that their expansion contains a relevant b-close co-occurrence of P_1, P_2. By Claim 16, $|\mathcal{N}| \leq \mathrm{occ}$.

▶ **Lemma 24.** *Assume that P_2 is not a substring of P_1. One can retrieve in $O(m + (1 + \mathrm{occ}) \log^3 N)$ time a set $\mathcal{N}' \supset \mathcal{N}$, $|\mathcal{N}'| = O(\mathrm{occ} \log N)$.*

Proof. We start by computing $\mathrm{Splits}'(G', P_1)$ and $\mathrm{Splits}'(G', P_2)$ via Lemma 8 in $O((|P_1| + |P_2|) \log N) = O(m \log N)$ time (or providing a certificate that either P_1 or P_2 does not occur in S, in which case there are no co-occurrences of P_1, P_2 in S and we are done). Recall that $|\mathrm{Splits}'(G', P_1)|, |\mathrm{Splits}'(G', P_2)| \in O(\log N)$. For each fixed pair of splits $s_1 \in \mathrm{Splits}'(G', P_1)$, $s_2 \in \mathrm{Splits}'(G', P_2)$ and $j \in \{1, 2\}$, we compute the interval of strings in T_{pre} prefixed by $\mathrm{rev}(P_j[\dots s_j])$, which corresponds to the locus u_j of $\mathrm{rev}(P_j[\dots s_j])$ in T_{pre} and the interval of strings in T_{suf} prefixed by $P_j(s_j \dots]$, which corresponds to the locus v_j of $P_j(s_j \dots]$ in T_{suf}. Computing the intervals takes $O(m + \log^2 N)$ time for all the splits by Lemma 9.

Consider the strings $S_1 = \text{rev}(U_1)V_1$ and $S_2 = \text{rev}(U_2)V_2$, where U_1, U_2, V_1, V_2 are the labels of u_1, v_1, u_2, v_2, respectively. Let $l_1 = |\text{rev}(U_1)|$, $\Delta_1 = l_1 - s_1$, $l_2 = |\text{rev}(U_2)|$, $\Delta_2 = l_2 - s_2$, and $\Delta = \Delta_1 - \Delta_2$.

Consider a relevant co-occurrence (q_1, q_2) of P_1, P_2 in the expansion of a non-terminal A. By Corollary 19, q_1, q_2 imply existence of occurrences p_1, p_2 of S_1, S_2 such that $[p_1, p_1 + |S_1|) \supseteq [q_1, q_1 + |P_1|)$ and $[p_2, p_2 + |S_2|) \supseteq [q_2, q_2 + |P_2|)$. Our index must treat both cases of Corollary 19. We consider eight subcases defined in Fig. 3, which describe all possible locations of p_1 and p_2.

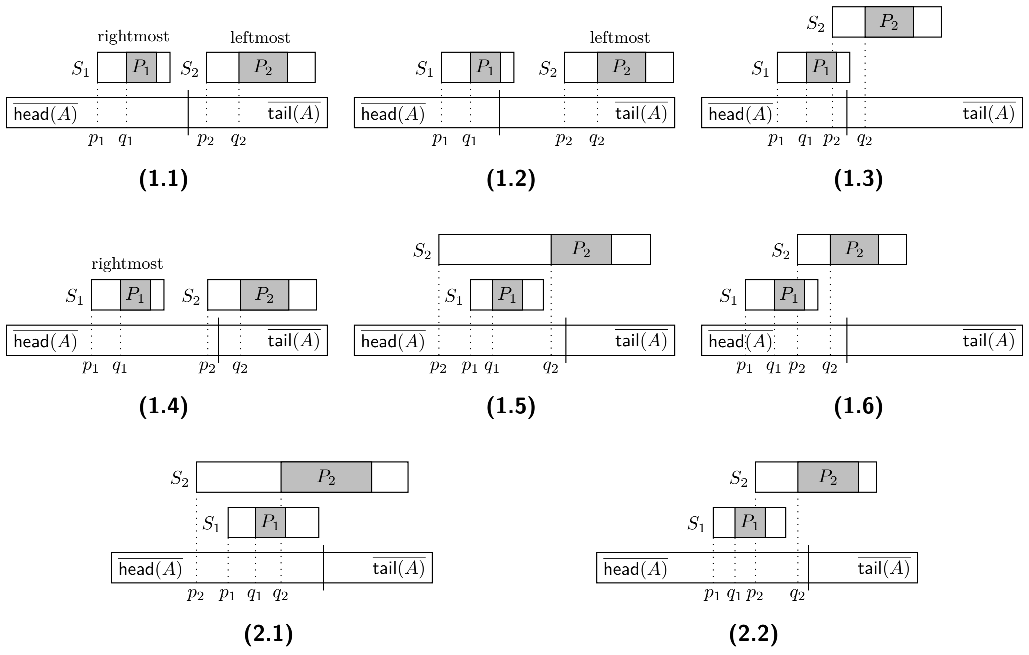


Figure 3 Assume that S_1 does not contain S_2. The figure shows all possible locations of occurrences p_1, p_2 of S_1, S_2 in $\overline{A}$. **In Case 1 of Corollary 19**, there are six subcases: (1.1) $p_1 + |S_1| - 1 \leq |\overline{\text{head}(A)}|$, $p_2 > |\overline{\text{head}(A)}|$; (1.2) p_1 is a relevant occurrence of S_1, $p_2 > |\overline{\text{head}(A)}|$; (1.3) p_1, p_2 are relevant; (1.4) p_2 is relevant, $p_1 + |S_1| - 1 \leq p_2$; (1.5) p_2 is relevant, $p_2 < p_1 \leq p_1 + |S_1| - 1 \leq p_2 + |S_2| - 1$; (1.6) p_2 is relevant, $p_1 < p_2 < p_1 + |S_1| - 1 \leq p_2 + |S_2| - 1$. By the definition of a co-occurrence and by Observation 20, in Subcases (1.1) and (1.4) p_1 must be as far to the right as possible, and in Subcases (1.1) and (1.2) p_2 must be as far to the left as possible. **In Case 2**, there are two subcases: (2.1) p_2 is relevant and $p_2 \leq p_1 \leq p_1 + |S_1| - 1 \leq p_2 + |S_2| - 1$; (2.2) p_2 is relevant and $p_1 < p_2 < p_2 + \pi_1 - 1 \leq p_1 + |S_1| - 1$, where π_1 is the period of S_1. In all subcases, $q_2 = p_2 + \Delta_2$. In Subcases (1.1)-(1.6) $q_1 = p_1 + \Delta_1$ and in Subcases (2.1) and (2.2) $q_1 = p_1 + \Delta_1 + k \cdot \pi_1$ for some integer k.

Subcases (1.1)–(1.4). To retrieve the non-terminals, we query $\mathcal{T}_1(u_1, u_2, v_1, v_2)$ to find all integers that belong to the range $[\Delta, \Delta + b]$ (and the corresponding non-terminals). Recall that, for each non-terminal A, the tree stores an integer $d = p_2 - p_1$, where p_1 is the starting position of an occurrence of S_1 in $\overline{A}$ and p_2 of S_2. By Observation 20, $p_1 + \Delta_1$ is an occurrence of P_1 and $p_2 + \Delta_2$ is an occurrence of P_2. The distance between them is in $[0, b]$ iff $d \in [\Delta, \Delta + b]$. By Observation 21, each retrieved non-terminal contains a close co-occurrence of (q_1, q_2). On other other hand, if $\overline{A}$ contains a co-occurrence (q_1, q_2) corresponding to one Subcases (1.1)-(1.4), then by Corollary 19, $p_1 = q_1 - \Delta_1$ is an occurrence of S_1 and $p_2 = q_2 - \Delta_2$ is an occurrence of S_2 and by construction $\mathcal{T}_1(u_1, u_2, v_1, v_2)$ stores an integer $d = p_2 - p_1$. Therefore, the query retrieves all non-terminals corresponding to Subcases (1.1)-(1.4).

Subcases (1.5) and (2.1). We must decide whether an occurrence of P_1 in S_2 forms a b-close co-occurrence with the occurrence Δ_2 of P_2 in S_2, and if so, report all non-terminals such that their expansion contains a relevant co-occurrence of S_2 with a split l_2, which are exactly the non-terminals stored in the list $\mathcal{L}(u_2, v_2)$. Let $k = \lceil \log(s_2) \rceil$. Recall that the index stores the following information for k:

1. p_1, the rightmost occurrence of S_1 in S_2 such that $p_1 + (|S_1| - 1) \le l_2 - 2^k$;
2. p_1', the leftmost occurrence of S_1 in S_2 such that $p_1' \le l_2 - 2^k \le p_1 + (|S_1| - 1)$;
3. p_1'', the rightmost occurrence of S_1 in S_2 such that $p_1'' \le l_2 - 2^k \le p_1'' + (|S_1| - 1)$.

(See Fig. 4). By Observation 20, the occurrence p_1 of S_1 induces an occurrence $q_1 = p_1 + \Delta_1$ of P_1. Furthermore, if S_1 is periodic with period π_1, then $q_1 + \pi_1 \cdot k$, $0 \le k \le \lfloor (|S_1| - q_1 - |P_1|)/\pi_1 \rfloor$, are also occurrences of P_1. One can decide whether the distance from any of these occurrences to q_2 is in $[0, b]$ in constant time, and if yes, then there S_2 contains a b-close co-occurrence of P_1, P_2 by Observation 21. Second, by Corollary 2, if S_1 is not periodic, then there are no occurrences of S_1 between p_1' and p_1'' and p_1', p_1'' by Observation 20 induce occurrences $p_1' + \Delta_1, p_1'' + \Delta_1$ of P_1. Otherwise, there are occurrences of P_1 in every position $p_1' + \Delta_1 + k \cdot \pi_1$, $0 \le k \le \lfloor (|S_1| + p_1'' - |P_1| - p_1')/\pi_1 \rfloor$. Similarly, we can decide whether the distance from any of them to the occurrence Δ_2 of P_2 in S_2 is in $[0, b]$ in constant time. Finally, let q_1 be the rightmost occurrence of P_1 in S_2 in the interval $[l_2 - 2^k + 1, \Delta_2]$. We extract $S_2(l_2 - 2^k, \Delta_2 + |P_2|)$ via Fact 28 and search for q_1 using a linear-time pattern matching algorithm for P_1, which takes $O(|P_1| + |P_2|) = O(m)$ time. If $0 \le \Delta_2 - q_1 \le b$, then there is a b-close co-occurrence of P_1, P_2 in S_2. Correctness follows from Corollary 19, Observation 20 and Observation 21.

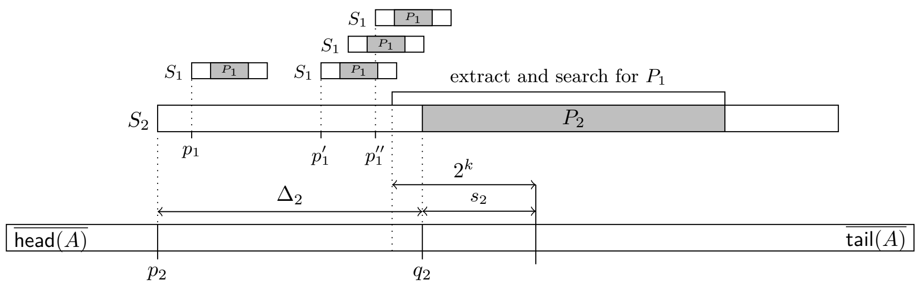


Figure 4 Query algorithm for Subcases (1.5) and (2.1).

Subcase (2.2). Let π_1 be the period of S_1. We retrieve the non-terminals associated with the integers $q \in \mathcal{T}_2(u_1, u_2, v_1, v_2)$ such that the intersection of an interval $I = [a, b]$ and $[\ell, q]$ is non-empty, where $a = \lceil (\Delta - \text{ov}(S_1, S_2))/\pi_1 \rceil$, $b = \lfloor (\Delta - \text{ov}(S_1, S_2) + b)/\pi_1 \rfloor$ and $\ell = -\lfloor (|S_1| - |P_1| - \Delta_1)/\pi_1 \rfloor$ (See the description of the index for the definition of $\text{ov}(S_1, S_2)$). As ℓ is fixed, we can implement the query via at most one binary tree search: If $b \le \ell$, the output is empty, if $a \le \ell \le b$, we must output all integers, and if $\ell \le a$, we must output all $q \ge b$. Let us now explain why the algorithm is correct. Consider a non-terminal A for which $\mathcal{T}_2(u_1, u_2, v_1, v_2)$ stores an integer q. By construction, $\overline{A}$ contains a relevant occurrence of S_2 with a split l_2. A position $p_1 = |\overline{\text{head}(A)}| - l_2 - \text{ov}(S_1, S_2) - q \cdot \pi_1$ is the leftmost occurrence of S_1 in $\overline{A}$ such that $p_1 \le p_2 \le p_1 + |S_1| - 1$ and $p_2 = |\overline{\text{head}(A)}| - l_2 - \text{ov}(S_1, S_2)$ the rightmost. Consequently, there is an occurrence $q_1 = |\overline{\text{head}(A)}| - l_2 - \text{ov}(S_1, S_2) - q' \cdot \pi_1 + \Delta_1$ of P_1 for each $-\lfloor (|S_1| - |P_1| - \Delta_1)/\pi_1 \rfloor \le q' \le q$. The occurrence of S_2 implies that $q_2 = |\overline{\text{head}(A)}| - s_2$ is an occurrence of P_2. We have $0 \le q_2 - q_1 = q' \cdot \pi_1 + \text{ov}(S_1, S_2) - \Delta \le b$ iff $\Delta - \text{ov}(S_1, S_2) \le q' \cdot \pi_1 \le \Delta - \text{ov}(S_1, S_2) + b$, which is equivalent to $[\ell, q] \cap I \ne \emptyset$. It follows that we retrieve every non-terminal corresponding to Subcase (2.2). On the other hand, by Observation 21, the expansion of each retrieved non-terminal contains a b-close co-occurrence of P_1, P_2.

Subcase (1.6). We argue that we have already reported all non-terminals corresponding to this subcase and there is nothing left to do. Consider a non-terminal A such that its expansion contains a relevant occurrence p_2 of S_2. If there are at most two occurrences p_1 of S_1 such that $p_1 \le p_2 \le p_1 + |S_1| - 1 \le p_2 + |S_2| - 1$, we will treat them when we query $\mathcal{T}_1(u_1, u_2, v_1, v_2)$ (Subcases (1.1)-(1.4)). Otherwise, by Corollary 2, S_1 is periodic and there is an occurrence p_1' of S_1 such that $p_1' \le p_2 < p_2 + \pi_1 \le p_1 + |S_1| - 1 < p_2 + |S_2| - 1$. The non-terminals corresponding to this case are reported when we query $\mathcal{T}_2(u_1, u_2, v_1, v_2)$ (Subcase (2.2)).

Time complexity. As shown above, the algorithm reports a set $\mathcal{N}' \supset \mathcal{N}$ of non-terminals and each non-terminal in $\mathcal{N}'$ contains a b-close co-occurrence. By Claim 16 and since the height of G' is $h = O(\log N)$, we have $|\mathcal{N}'| = O(\text{occ} \log N)$. Furthermore, for a fixed pair of splits of P_1, P_2, each non-terminal in $\mathcal{N}'$ can be reported a constant number of times. Since $|\text{Splits}'(G', P_1)| \cdot |\text{Splits}'(G', P_2)| = O(\log^2 N)$, the total size of the output is $|\mathcal{N}'| \cdot O(\log^2 N) = O(\text{occ} \cdot \log^3 N)$. We therefore obtain that the running time of the algorithm is $O(m + \log^3 N + \text{occ} \log^3 N) = O(m + (1 + \text{occ}) \log^3 N)$ as desired. ◀

Once we have retrieved the set $\mathcal{N}'$, we find all b-close relevant co-occurrences for each of the non-terminals in $\mathcal{N}'$ using Theorem 10. In fact, our algorithm acts naively and computes *all* relevant co-occurrences for a non-terminal in $\mathcal{N}'$, and then selects those that are b-close. By case inspection, one can show that a relevant co-occurrence for a non-terminal A always consists of an occurrence of P_2 that is either relevant or the leftmost in $\overline{\mathsf{tail}(A)}$, and a preceding occurrence of P_1. Intuitively, this allows to compute all relevant co-occurrences efficiently and guarantees that their number is small. Formally, we show the following claim:

► **Lemma 25.** *Assume that P_2 is not a substring of P_1. After $O(m \log N + \log^2 N)$-time preprocessing, the data structure of Theorem 10 allows to compute all b-close relevant co-occurrences of P_1, P_2 in the expansion of a given non-terminal A in time $O(\log^3 N \log\log N)$.*

A part of the index of Christiansen et al. [8] is a pruned copy of the parse tree of G'. They showed how to traverse the tree to report all occurrences of a pattern, given its relevant occurrences in the non-terminals. By using essentially the same algorithm, we can report all b-close co-occurrences in amortized constant time per co-occurrence, which concludes the proof of Theorem 1. (See Appendix C, Lemma 33.)

References

1 Amir Abboud, Arturs Backurs, Karl Bringmann, and Marvin Künnemann. Fine-grained complexity of analyzing compressed data: Quantifying improvements over decompress-and-solve. In *Proc. 58th FOCS*, pages 192–203, 2017.

2 Amir Abboud, Arturs Backurs, Karl Bringmann, and Marvin Künnemann. Impossibility results for grammar-compressed linear algebra. In *Proc. 34th NeurIPS*, pages 8810–8823, 2020.

3 Djamal Belazzougui, Paolo Boldi, Rasmus Pagh, and Sebastiano Vigna. Fast prefix search in little space, with applications. In *Proc. 18th ESA*, pages 427–438, 2010.

4 Philip Bille, Inge Li Gørtz, Max Rishøj Pedersen, and Teresa Anna Steiner. Gapped indexing for consecutive occurrences. In *Proc. 32nd CPM*, pages 10:1–10:19, 2021.

5 Philip Bille, Gad M. Landau, Rajeev Raman, Kunihiko Sadakane, Srinivasa Rao Satti, and Oren Weimann. Random access to grammar-compressed strings and trees. *SIAM J. Comput.*, 44(3):513–539, 2015.

6 Timothy M. Chan. Persistent predecessor search and orthogonal point location on the word RAM. *ACM Trans. Algorithms*, 9(3):22:1–22:22, 2013.

7 Moses Charikar, Eric Lehman, Ding Liu, Rina Panigrahy, Manoj Prabhakaran, April Rasala, Amit Sahai, and Abhi Shelat. Approximating the smallest grammar: Kolmogorov complexity in natural models. In *Proc. 34th STOC*, pages 792–801, 2002.

8 Anders Roy Christiansen, Mikko Berggren Ettienne, Tomasz Kociumaka, Gonzalo Navarro, and Nicola Prezza. Optimal-time dictionary-compressed indexes. *ACM Trans. Algorithms*, 17(1):8:1–8:39, 2021.

9 Francisco Claude and Gonzalo Navarro. Improved grammar-based compressed indexes. In *Proc. 19th SPIRE*, pages 180–192, 2012.

10 Francisco Claude, Gonzalo Navarro, and Alejandro Pacheco. Grammar-compressed indexes with logarithmic search time. *J. Comput. Syst. Sci.*, 118:53–74, 2021.

11 Maxime Crochemore. Constant-space string-matching. In *Proc. 8th FSTTCS*, pages 80–87, 1988.

12 Diego Díaz-Domínguez, Gonzalo Navarro, and Alejandro Pacheco. An LMS-based grammar self-index with local consistency properties. In *Proc. 28th SPIRE*, 2021.

13 Paolo Ferragina and Rossano Venturini. Indexing compressed text. In *Encyclopedia of Database Systems (2nd ed.)*. Springer, 2018.

14 Nathan J. Fine and Herbert S. Wilf. Uniqueness theorems for periodic functions. *Proc. Am. Math. Soc.*, 16(1):109–114, 1965.

15 Johannes Fischer, Travis Gagie, Tsvi Kopelowitz, Moshe Lewenstein, Veli Mäkinen, Leena Salmela, and Niko Välimäki. Forbidden patterns. In *Proc. 10th LATIN*, pages 327–337, 2012.

16 Travis Gagie, Pawel Gawrychowski, Juha Kärkkäinen, Yakov Nekrich, and Simon J. Puglisi. LZ77-based self-indexing with faster pattern matching. In *Proc. 11th LATIN*, pages 731–742, 2014.

17 Travis Gagie, Gonzalo Navarro, and Nicola Prezza. Optimal-time text indexing in BWT-runs bounded space. In *Proc. 29th SODA*, pages 1459–1477, 2018.

18 Pawel Gawrychowski, Adam Karczmarz, Tomasz Kociumaka, Jakub Lacki, and Piotr Sankowski. Optimal dynamic strings. In *Proc. 29th SODA*, pages 1509–1528, 2018.

19 Daniel Gibney and Sharma V. Thankachan. Text indexing for regular expression matching. *Algorithms*, 14(5):133, 2021.

20 Leszek Gąsieniec, Roman M. Kolpakov, Igor Potapov, and Paul Sant. Real-time traversal in grammar-based compressed files. In *Proc. 15th DCC*, page 458, 2005.

21 Dan Gusfield. *Algorithms on Strings, Trees, and Sequences - Computer Science and Computational Biology*. Cambridge University Press, 1997.

22 Wing-Kai Hon, Rahul Shah, Sharma V. Thankachan, and Jeffrey Scott Vitter. String retrieval for multi-pattern queries. In *Proc. 17th SPIRE*, pages 55–66, 2010.

23 Wing-Kai Hon, Rahul Shah, Sharma V. Thankachan, and Jeffrey Scott Vitter. Document listing for queries with excluded pattern. In *Proc. 23rd CPM*, pages 185–195, 2012.

24 Richard M Karp and Michael O Rabin. Efficient randomized pattern-matching algorithms. *IBM J. Res. Dev*, 31(2):249–260, 1987.

25 John C. Kieffer and En-Hui Yang. Grammar-based codes: A new class of universal lossless source codes. *IEEE Trans. Inf. Theory*, 46(3):737–754, 2000.

26 Tsvi Kopelowitz and Robert Krauthgamer. Color-distance oracles and snippets. In *Proc. 27th CPM*, pages 24:1–24:10, 2016.

27 Tsvi Kopelowitz, Seth Pettie, and Ely Porat. Higher lower bounds from the 3SUM conjecture. In *Proc. 27th SODA*, pages 1272–1287, 2016.

28 Kasper Green Larsen, J. Ian Munro, Jesper Sindahl Nielsen, and Sharma V. Thankachan. On hardness of several string indexing problems. *Theor. Comput. Sci.*, 582:74–82, 2015.

29 Moshe Lewenstein. Orthogonal range searching for text indexing. In *Space-Efficient Data Structures, Streams, and Algorithms*, pages 267–302, 2013.
30 Veli Mäkinen and Gonzalo Navarro. Compressed text indexing. In *Encyclopedia of Algorithms*, pages 394–397. Springer New York, 2016.
31 Udi Manber and Eugene W. Myers. Suffix arrays: A new method for on-line string searches. *SIAM J. Comput.*, 22(5):935–948, 1993.
32 S. Muthukrishnan. Efficient algorithms for document retrieval problems. In *Proc. 13th SODA*, pages 657–666, 2002.
33 Gonzalo Navarro and Sharma V. Thankachan. Reporting consecutive substring occurrences under bounded gap constraints. *Theor. Comput. Sci.*, 638:108–111, 2016.
34 Takaaki Nishimoto, Tomohiro I, Shunsuke Inenaga, Hideo Bannai, and Masayuki Takeda. Fully dynamic data structure for LCE queries in compressed space. In *Proc. 41st MFCS*, volume 58, pages 72:1–72:15, 2016.
35 Wojciech Rytter. Application of Lempel-Ziv factorization to the approximation of grammar-based compression. *Theor. Comput. Sci.*, 302(1-3):211–222, 2003.
36 Peter Weiner. Linear pattern matching algorithms. In *Proc. 14th SWAT*, pages 1–11, 1973.

A Proofs omitted from Section 2

▶ **Lemma 9.** *Given an RLSLP G of size g and height h. Assume that every string in a set $\mathcal{S}$ is either a prefix or a suffix of the expansion of a non-terminal of G or its reverse. The trie for $\mathcal{S}$ can be implemented in space $O(|\mathcal{S}|)$ to maintain the following queries in $O(m + \tau \cdot (h + \log m))$ time: Given a pattern P of length m and suffixes Q_i of P, $1 \le i \le \tau$, find, for each i, the interval of strings in the (lexicographically sorted) $\mathcal{S}$ prefixed by Q_i.*

Proof. Let us first recall the definition of the Karp–Rabin fingerprint.

▶ **Definition 26** (Karp–Rabin fingerprint). *For a prime p and an $r \in \mathbb{F}_p^*$, the* Karp–Rabin fingerprint *[24] of a string X is defined as a tuple $(r^{|X|-1} \mod p, r^{-|X|+1} \mod p, \varphi_{p,r}(X))$, where $\varphi_{p,r}(X) = \sum_{k=0}^{|X|-1} S[k] r^k \mod p$.*

We use the following result of Christiansen et al. [8], which builds on Belazzougui et al. [3] and Gagie et al. [16, 17].

▶ **Fact 27** ([8, Lemma 6.5]). *Let $\mathcal{S}$ be a set of strings and assume we have a data structure supporting extraction of any length-l prefix of strings in $\mathcal{S}$ in time $f_e(l)$ and computing the Karp–Rabin fingerprint φ of any length-l prefix of a string in $\mathcal{S}$ in time $f_h(l)$. We can then build a data structure that uses $O(|\mathcal{S}|)$ space and supports the following queries in $O(m + f_e(m) + \tau(f_h(m) + \log m))$ time: Given a pattern P of length m and $\tau > 0$ suffixes $Q_1, \ldots, Q_\tau$ of P, find the intervals of strings in (the lexicographically-sorted) $\mathcal{S}$ prefixed by $Q_1, \ldots, Q_\tau$.*

It should be noted that despite using a hash function, the query algorithm is deterministic: the proof shows that p and r can be chosen during the construction time to ensure that there are no collisions on the substrings of the strings in $\mathcal{S}$.

To bound f_e, we use [8, Lemma 6.6] which builds on Gąsieniec et al. [20] and Claude and Navarro [9].

▶ **Fact 28** ([8, Lemma 6.6]). *Given an RLSLP of size $O(g)$, there exists a data structure of size $O(g)$ such that any length-l prefix or suffix of $\overline{A}$ can be obtained from any non-terminal A in time $f_e(l) = O(l)$.*

To bound $f_h(l)$, we introduce a simple construction based on the following well-known fact:

▶ **Fact 29.** *Consider strings X, Y, Z where $XY = Z$. Given the Karp–Rabin fingerprints of two of the three strings, one can compute the fingerprint of the third string in constant time.*

▷ Claim 30. Given a RLSLP G of size g and height h, there exists a data structure of size $O(g)$ that given a non-terminal A and an integer l allows to retrieve the Karp-Rabin fingerprints of the length-l prefix and suffix of $\overline{A^r}$ and $\text{rev}(\overline{A^r})$ in time $f_h(l) = O(h + \log l)$.

Proof. The claim for $\text{rev}(\overline{A^r})$ follows for the claim for $\overline{A^r}$ by considering the grammar G_{rev}, where the order of the non-terminals in each production is reversed. Below we focus on extracting the fingerprints for $\overline{A^r}$, and we further restrict our attention to prefixes of $\overline{A^r}$, the algorithm for suffixes being analogous.

The data structure consists of two sets. The first set contains the lengths of the expansions of all non-terminals in the grammar, and the second one their fingerprints.

By Fact 29 and doubling, it suffices to show an algorithm for computing the fingerprint of the length-l prefix of $\overline{A}$. Assume that A associated with a rule $A \to BC$. If the length of $\overline{A}$ is smaller than l, we return error. Otherwise, to compute the fingerprint of the length-l prefix of $\overline{A}$, we consider two cases. If $l \le |\overline{B}|$, we recurse on B to retrieve the fingerprint of the l-length prefix of $\overline{B}$. Otherwise, we recurse on C to retrieve the fingerprint of $\overline{C}[\dots l - |\overline{B}|)$ and then compute the fingerprint of the l-length prefix of $\overline{A}$ from the fingerprints of $\overline{B}$ and $\overline{C}[\dots l - |\overline{B}|)$ in constant time by Fact 29.

For a non-terminal A associated with a rule $A \to B^r$, we compute the fingerprint analogously. If the length of $\overline{A}$ is smaller than l, we return error. Otherwise, let q be such that $q \cdot |\overline{B}| \le l < (q+1) \cdot |\overline{B}|$. We compute the fingerprint of $\overline{B}^q$ from the fingerprint of $\overline{B}$ by applying Fact 29 $O(1 + \log q)$ times, and the fingerprint of $\overline{B}[\dots l - q \cdot |\overline{B}|)$ recursively. We can then apply Fact 29 to compute the fingerprint of the length-l prefix of $\overline{A}$ in constant time. Note that in this case, the length of the prefix decreases by a factor at least q.

If we are in a terminal A, the calculation takes $O(1)$ time (the prefix must be equal to A itself).

In total, we spend $O(h + \log l)$ time as we recurse $O(h)$ times, and whenever we spend more than constant time in a symbol, we charge it on the decrease in the length. The fingerprints of length-l suffixes are computed analogously. ◁

By substituting the bounds for $f_e(l)$ (Fact 29) and $f_h(l)$ (Claim 28) into Fact 27, we obtain the claim of the lemma. ◀

B Proofs omitted from Section 3

▷ Claim 31. Given a non-terminal A of G', we can find the leftmost and the rightmost occurrences of P in $\overline{A}$ and as a corollary in $\overline{\mathsf{head}(A)}$ and $\overline{\mathsf{tail}(A)}$ in $O(\log^2 N \log\log N)$ time.

Proof. We explain how to find the leftmost occurrence of P in $\overline{A}$, the rightmost one can be found analogously. We first check whether $\overline{A}$ contains an occurrence of P via Claim 14 in $O(\log N \log\log N)$ time. If it does not, we can stop immediately. Below we assume that there is an occurrence of P in $\overline{A}$. Next, we check whether $\overline{\mathsf{head}(A)}$ contains an occurrence of P via Claim 14 in $O(\log N \log\log N)$ time. If it does, the leftmost occurrence of P in $\overline{A}$ is the leftmost occurrence of P in $\overline{\mathsf{head}(A)}$ and we can find it by recursing on $\mathsf{head}(A)$. If $\overline{\mathsf{head}(A)}$ does not contain an occurrence of P, but $\overline{A}$ contains relevant occurrences of P, then the leftmost occurrence of P in $\overline{A}$ is the leftmost relevant occurrence of P in $\overline{A}$ and we can find it in $O(|\text{Splits}'(G', P)|) = O(\log N)$ time. Finally, if P neither occurs in $\overline{\mathsf{head}(A)}$ nor has relevant occurrences in $\overline{A}$, then the leftmost occurrence of P in $\overline{A}$ is the leftmost occurrence of P in $\overline{\mathsf{tail}(A)}$. If $\mathsf{tail}(A)$ is a non-terminal C, we recurse on C to find it. If $\mathsf{tail}(A) = B^{r-1}$ for a non-terminal B, $\overline{\mathsf{tail}(A)}$ cannot contain an occurrence of P because $\overline{B}$ does not contain P and there are no relevant occurrences in A. We recurse down at most $h = O(\log N)$ levels, and spend $O(\log N \log\log N)$ time per level. The claim follows. ◁

▶ **Lemma 32.** *Let A be a non-terminal of G'. For any position p, we can find the rightmost occurrence $q \le p$ of P in $\overline{A}$ and the leftmost occurrence $q' \ge p$ of P in $\overline{A}$ in $O(\log^3 N \log\log N)$ time.*

Proof. First we describe how to locate q. Consider a node u of the parse tree of G' labeled by A. The algorithm starts at u and recurses down. Let A' be the label of the current node. It computes the leftmost and rightmost occurrences in $\overline{A'}$, $\overline{\mathsf{head}(A')}$ and $\overline{\mathsf{tail}(A')}$ as well as all relevant occurrences via Claim 31. If the leftmost occurrence of P in $\overline{A'}$ is larger than p, the search result is empty. Otherwise, consider two cases.

1. A' is associated with a rule $A' \to B'C'$, i.e. $\mathsf{head}(A') = B'$, $\mathsf{tail}(A') = C'$.
   a. If $p \le |\overline{B'}|$, recurse on B'.
   b. Assume now that $p > |\overline{B'}|$. If the leftmost occurrence of P in $\overline{C'}$ is smaller than p, recurse on C'. Otherwise, return the rightmost relevant occurrence of P in $\overline{A'}$ if it exists else the rightmost occurrence of P in $\overline{B'}$.
2. A' is associated with a rule $A \to (B')^r$, i.e. $\mathsf{head}(A') = B'$, $\mathsf{tail}(A') = (B')^{r-1}$. Let an integer k be such that $(k-1) \cdot |\overline{B'}| + 1 \le p \le k \cdot |\overline{B'}|$. The desired occurrence of P is the rightmost one of the following ones:
   a. The rightmost occurrence $q \le p$ of P which crosses the border between two copies of $\overline{B'}$. To compute q, we compute all relevant occurrences of P in $\overline{A'}$ and then shift each of them by the maximal possible shift $r' \cdot |\overline{B'}|$, where r' is an integer, which guarantees that it starts before p and ends before $|\overline{A'}|$ and take the rightmost of the computed occurrences to obtain q.
   b. The rightmost occurrence q of P such that for some integer k', we have $(k'-1) \cdot |\overline{B'}| \le q \le q + |P| - 1 \le k' \cdot |\overline{B'}|$ (i.e. the occurrence fully belongs to some copy of $\overline{B'}$). In this case, q is either the rightmost occurrence of P in the $(k-1)$-th copy of $\overline{B'}$, or the rightmost occurrence of P in the k-th copy of $\overline{B'}$ that is smaller than p. In the second case, we compute q by recursing on B'.

We recurse down at most h levels. On each level we spend $O(\log^2 N \log\log N)$ time to compute the leftmost, the rightmost, and relevant occurrences and respective shifts for a constant number of non-terminals via Claim 31. Therefore, in total we spend $O(h \cdot \log^2 N \log\log N) = O(\log^3 N \log\log N)$ time.

Locating q' is very similar and differs only in small technicalities. The algorithm starts at the node u and recurses down. Let A' be the label of the current node. We compute the leftmost and rightmost occurrences in $\overline{A'}$, $\overline{\mathsf{head}(A')}$ and $\overline{\mathsf{tail}(A')}$ as well as all relevant occurrences via Claim 31. If the rightmost occurrence of P in $\overline{A'}$ is smaller than p, the search result is empty. Otherwise, consider two cases.

1. A' is associated with a rule $A' \to B'C'$, i.e. $\mathsf{head}(A') = B'$, $\mathsf{tail}(A') = C'$.
   a. If $p > |\overline{B'}|$, recurse on C'.
   b. Assume now that $p \le |\overline{B'}|$. If the rightmost occurrence of P in $\overline{B'}$ is larger than p, recurse on B'. Otherwise, return the leftmost relevant occurrence q satisfying $q \ge p$, if it exists, and otherwise the leftmost occurrence of P in $\overline{C'}$.

2. A' is associated with a rule $A \to (B')^r$, i.e. $\mathsf{head}(A') = B'$, $\mathsf{tail}(A') = (B')^{r-1}$. Let an integer k be such that $(k-1) \cdot |\overline{B'}| + 1 \le p \le k \cdot |\overline{B'}|$. The desired occurrence of P is the leftmost one of the following ones:
   a. The leftmost occurrence $q' \ge p$ of P which crosses the border between two copies of $\overline{B'}$. To compute q', we compute all relevant occurrences of P in $\overline{A'}$ and then shift each of them by the minimal possible shift $r' \cdot |\overline{B'}|$, where r' is an integer, which guarantees that it starts after p and ends before $|\overline{A'}|$ (if it exists) and take the leftmost of the computed occurrences to obtain q.
   b. The leftmost occurrence q' of P such that for some integer k', we have $(k'-1) \cdot |\overline{B'}| \le q' \le q' + |P| - 1 \le k' \cdot |\overline{B'}|$ (i.e. the occurrence fully belongs to some copy of $\overline{B'}$). In this case, q' is either the leftmost occurrence of P in the $(k+1)$-st copy of $\overline{B'}$, or the leftmost occurrence of P in the k-th copy of $\overline{B'}$ that is larger than p. In the second case, we compute q' by recursing on B'.

The time complexities are the same as for computing q. ◀

C Proofs omitted from Section 4

▶ **Lemma 25.** *Assume that P_2 is not a substring of P_1. After $O(m \log N + \log^2 N)$-time preprocessing, the data structure of Theorem 10 allows to compute all b-close relevant co-occurrences of P_1, P_2 in the expansion of a given non-terminal A in time $O(\log^3 N \log\log N)$.*

Proof. We preprocess P_1, P_2 in $O(m \log N + \log^2 N)$ time as explained in Theorem 10. Upon receiving a non-terminal A, we compute the leftmost and the rightmost occurrences of P_1, P_2 in $\overline{\mathsf{head}(A)}$ and $\overline{\mathsf{tail}(A)}$, as well as a set Π_1 of all relevant occurrences of P_1 in $\overline{A}$ and a set Π_2 of all relevant occurrences of P_2 in $\overline{A}$ via Claim 31. We will compute all relevant co-occurrences in $\overline{A}$, selecting those of them that are b-close is then trivial. As $q_1 \le q_2$ by definition, each relevant co-occurrence (q_1, q_2) of P_1, P_2 in $\overline{A}$ falls under one of the following categories:

1. q_1 is a relevant occurrence of P_1 in $\overline{A}$ and q_2 is a relevant occurrence of P_2 in $\overline{A}$ (i.e. $q_1 \in \Pi_1, q_2 \in \Pi_2$). To check whether a pair $q_1 \in \Pi_1, q_2 \in \Pi_2$ forms a co-occurrence of P_1, P_2 in $\overline{A}$, we must check whether there is an occurrence q of either P_1 or P_2 between q_1 and q_2. The occurrence q can only be the rightmost occurrence r_q of P_2 in $\overline{\mathsf{head}(A)}$, the leftmost occurrence l_q of P_1 in $\overline{\mathsf{tail}(A)}$, or an occurrence in $\Pi_1 \cup \Pi_2$. Consequently, we can find all co-occurrences in this category by merging two (sorted) sets: $\Pi_1 \cup \{l_q\}$ and $\{r_q\} \cup \Pi_2$, which can be done in $O(2 + |\Pi_1 \cup \Pi_2|)$ time.
2. $1 \le q_1 \le q_1 + |P_1| - 1 \le |\overline{\mathsf{head}(A)}|$ and $|\overline{\mathsf{head}(A)}| < q_2 \le q_2 + |P_2| - 1$. In this case, q_1 must be the rightmost occurrence of P_1 in $\overline{\mathsf{head}(A)}$ and q_2 the leftmost occurrence in $\overline{\mathsf{tail}(A)}$, $q_1 \le q_2$, and there must be no occurrence $q \in \Pi_1 \cup \Pi_2$ such that $q_1 \le q \le q_2$. Therefore, if there is a co-occurrence in this category, we can retrieve it in $O(|\Pi_1 \cup \Pi_2|)$ time.
3. q_1 is a relevant occurrence of P_1 in $\overline{A}$ (i.e. $q_1 \in \Pi_1$) and $|\overline{\mathsf{head}(A)}| < q_2 \le q_2 + |P_2| - 1$. In this case, q_1 must be the rightmost occurrence in Π_1 and q_2 the leftmost occurrence of P_2 in $\overline{\mathsf{tail}(A)}$, and there should be no occurrence from Π_2 between q_1 and q_2. Therefore, if there is a co-occurrence in this category, we can find it in $O(|\Pi_1 \cup \Pi_2|)$ time.
4. $q_1 \le q_1 + |P_1| - 1 \le |\overline{\mathsf{head}(A)}|$ and q_2 is a relevant occurrence of P_2 in $\overline{A}$ (i.e. $q_2 \in \Pi_2$). First, consider the leftmost occurrence in $q_2 \in \Pi_2$. We find the rightmost occurrence $q_1 \le q_2$ of P_1 in $\overline{A}$ via a predecessor query. The pair (q_1, q_2) is a co-occurrence iff the rightmost occurrence of P_2 in $\overline{\mathsf{head}(A)}$ is smaller than q_1, which can be checked in constant time. Second, we consider the remaining occurrences in Π_2. Let q_2' be the leftmost one.

We begin by computing the preceding occurrence q_1' of P_1 via a predecessor query and if $q_2 \leq q_1'$, output the resulting co-occurrence. If $\Pi_2 = \{q_2, q_2'\}$, we are done. Otherwise, by Corollary 2, the occurrences in $\Pi_2 \setminus \{q_2\}$ form an arithmetic progression with difference equal to the period of P_2 (as all of them contain the position $|\overline{\mathsf{head}(A)}|$). Furthermore, as P_1 does not contain P_2, the occurrence of P_1 preceding q_2' belongs to the periodic region formed by the relevant occurrences of P_2. Therefore, all the remaining co-occurrences can be obtained from the co-occurrence for q_2' by shifting them by the period. In total, this step takes $O(|\Pi_2| + \log^3 N \log\log N)$ time. ◀

▶ **Lemma 33.** *Assume that P_2 is not a substring of P_1. One can compute all b-close co-occurrences of P_1, P_2 in S in time $O(m + (1 + \mathrm{occ}) \cdot \log^4 N \log\log N)$.*

Proof. During the preprocessing, we prune the parse tree: First, for each non-terminal B, all but the first node labeled by B in the preorder is converted into a leaf and its subtree is pruned. For each node v labeled by a non-terminal B, we store $\mathrm{anc}(v)$, the nearest ancestor u of v labeled by A such that u is the root or A labels more than one node in the pruned tree. Second, for every node labeled by a non-terminal A associated with a rule $A \rightarrow B^k$, we replace its $k-1$ rightmost children with a leaf labeled by B^{k-1}. We call the resulting tree *the pruned parse tree* and for each node v labeled by a non-terminal B store $\mathrm{next}(v)$, the next node labeled by B in preorder, if there is one. As every non-terminal labels at most one internal node of the pruned parse tree and every node has at most two children, it occupies $O(g')$ space.

When the algorithm of Lemma 24 outputs $A \in \mathcal{N}'$, we compute all relevant co-occurrences (q_1, q_2) in $\overline{A}$ in time $O(\log^3 N \log\log N)$ using Lemma 25 and select those which satisfy $q_2 - q_1 \leq b$.

Fix a b-close relevant co-occurrence (q_1, q_2) in $\overline{A}$. If A is associated with a rule $A \rightarrow BC$, construct a set $\mathrm{occ}(A) := \{(q_1, q_2)\}$, and otherwise if A is associated with a rule $A \rightarrow B^k$,

$$\mathrm{occ}(A) := \{(q_1 + i \cdot |\overline{B}|, q_2 + i \cdot |\overline{B}|) : 0 \leq i \leq \lfloor (|\overline{A}| - q_2 - |P_2| + 1)/|\overline{B}| \rfloor\}$$

Suppose that A labels nodes $v_1, v_2, \ldots, v_k$ of the unpruned parse tree of G' (by construction v_1 is not pruned and we assimilate it to the corresponding node in the pruned parse tree). If W is a set of co-occurrences, denote for brevity $W + \delta = \{(q_1 + \delta, q_2 + \delta) : (q_1, q_2) \in W\}$. Below we show an algorithm that generates a set $\mathcal{S} = \cup_i \mathrm{occ}(A) + \mathrm{off}(v_i)$ that contains all secondary b-close co-occurrences due to (q_1, q_2).

We traverse the pruned parse tree, while maintaining a priority queue. The queue is initialized to contain the first node in the preorder labeled by A together with $\mathrm{occ}(A)$. Until the priority queue is empty, pop a node v and a set W of co-occurrences of P_1, P_2 in the expansion of its label, and perform the following steps:

- **Reporting step:** If v is the root, report W;
- **Next node step:** If $\mathrm{next}(v)$ is defined, push $(\mathrm{next}(v), W + \mathrm{off}(\mathrm{next}(v)) - \mathrm{off}(v))$;
- **Sibling step:** If v is labeled by a non-terminal B and its sibling by B^k, for some integer k, then $W := \cup_{0 \leq i \leq k} W + i \cdot |\overline{B}|$
- **Ancestor step:** Push to the queue $(\mathrm{anc}(v), W + \mathrm{off}(\mathrm{anc}(v)) - \mathrm{off}(v))$.

By construction and as every node is connected with the root by a path of anc links, the algorithm generates each co-occurrence in $\mathcal{S}$ exactly once. The time complexity follows: The algorithm of Lemma 24 takes $O(m + (1 + \mathrm{occ}) \cdot \log^3 N)$ time; applying Lemma 25 to every non-terminal in $\mathcal{N}'$ takes $O(\mathrm{occ} \cdot \log^4 N \log\log N)$ time; and maintaining the queue and reporting the co-occurrences takes $O(\mathrm{occ})$ time as at every step we can charge the time needed to update the queue on newly created co-occurrences. ◀