The Complexity of the Co-occurrence Problem

Philip Bille, Inge Li Gørtz, and Tord Stordalen(✉)

Technical University of Denmark, DTU Compute, Kgs. Lyngby, Denmark
{phbi,inge,tjost}@dtu.dk

Abstract. Let S be a string of length n over an alphabet Σ and let Q be a subset of Σ of size $q \geq 2$. The *co-occurrence problem* is to construct a compact data structure that supports the following query: given an integer w return the number of length-w substrings of S that contain each character of Q at least once. This is a natural string problem with applications to, e.g., data mining, natural language processing, and DNA analysis. The state of the art is an $O(\sqrt{nq})$ space data structure that—with some minor additions—supports queries in $O(\log\log n)$ time [CPM 2021].

Our contributions are as follows. Firstly, we analyze the problem in terms of a new, natural parameter d, giving a simple data structure that uses $O(d)$ space and supports queries in $O(\log\log n)$ time. The preprocessing algorithm does a single pass over S, runs in expected $O(n)$ time, and uses $O(d+q)$ space in addition to the input. Furthermore, we show that $O(d)$ space is optimal and that $O(\log\log n)$-time queries are optimal given optimal space. Secondly, we bound $d = O(\sqrt{nq})$, giving clean bounds in terms of n and q that match the state of the art. Furthermore, we prove that $\Omega(\sqrt{nq})$ bits of space is necessary in the worst case, meaning that the $O(\sqrt{nq})$ upper bound is tight to within polylogarithmic factors. All of our results are based on simple and intuitive combinatorial ideas that simplify the state of the art.

Keywords: Strings · Data structures · Lower bounds

1 Introduction

We consider the *co-occurrence problem* which is defined as follows. Let S be a string of length n over an alphabet Σ and let Q be a subset of Σ of size $q \geq 2$. For two integers i and j where $1 \leq i \leq j \leq n$, let $[i, j]$ denote the discrete interval $\{i, i+1, \ldots, j\}$, and let $S[i, j]$ denote the substring of S starting at $S[i]$ and ending at $S[j]$. The interval $[i, j]$ is a *co-occurrence* of Q in S if $S[i, j]$ contains each character in Q at least once. The goal is to preprocess S and Q into a data structure that supports the query

Philip Bille and Inge Li Gørtz are supported by Danish Research Council grant DFF-8021-002498.

– $\mathtt{co}_{S,Q}(w)$: return the number of co-occurrences of Q in S that have length w, i.e., the number of length-w substrings of S that contain each character in Q at least once.

For example, let $\Sigma = \{\mathtt{A}, \mathtt{B}, \mathtt{C}, \mathtt{-}\}$, $Q = \{\mathtt{A}, \mathtt{B}, \mathtt{C}\}$ and

$$S = \underset{1}{\mathtt{-}}\ \underset{2}{\mathtt{-}}\ \underset{3}{\mathtt{-}}\ \underset{4}{\mathtt{-}}\ \underset{5}{\mathtt{B}}\ \underset{6}{\mathtt{C}}\ \underset{7}{\mathtt{-}}\ \underset{8}{\mathtt{A}}\ \underset{9}{\mathtt{C}}\ \underset{10}{\mathtt{C}}\ \underset{11}{\mathtt{B}}\ \underset{12}{\mathtt{-}}\ \underset{13}{\mathtt{-}}\ .$$

Then

– $\mathtt{co}_{S,Q}(3) = 0$, because no length-three substring contains all three characters A, B, and C.
– $\mathtt{co}_{S,Q}(4) = 2$, because both $[5, 8]$ and $[8, 11]$ are co-occurrences of Q.
– $\mathtt{co}_{S,Q}(8) = 6$, because all six of the length-eight substrings of S are co-occurrences of Q.

Note that only sublinear-space data structures are interesting. With linear space we can simply precompute the answer to $\mathtt{co}_{S,Q}(i)$ for each $i \in [0, n]$ and support queries in constant time.

This is a natural string problem with applications to, e.g., data mining, and a large amount of work has gone towards related problems such as finding frequent items in streams [6–8,10] and finding frequent sets of items in streams [1,3,4,9,11,12,16]. Furthermore, it is similar to certain string problems, such as episode matching [5] where the goal is to determine all the substrings of S that occur a certain number of times within a given distance from each other. Whereas previous work is mostly concerned with identifying frequent patterns either in the whole string or in a sliding window of fixed length, Sobel, Bertram, Ding, Nargesian and Gildea [14] introduced the problem of studying a given pattern across all window lengths (i.e., determining $\mathtt{co}_{S,Q}(i)$ for all i). They motivate the problem by listing potential applications such as training models for natural language processing (short and long co-occurrences of a set of words tend to represent respectively syntactic and semantic information), automatically organizing the memory of a computer program for good cache behaviour (variables that are used close to each other should be near each other in memory), and analyzing DNA sequences (co-occurrences of nucleotides in DNA provide insight into the evolution of viruses). See [14] for a more detailed discussion of these applications.

Our work is inspired by [14]. They do not consider fast, individual queries, but instead they give an $O(\sqrt{nq})$ space data structure from which they can determine $\mathtt{co}_{S,Q}(i)$ for each $i = 1, \ldots, n$ in $O(n)$ time. Supporting fast queries is a natural extension to their problem, and we note that their solution can be extended to support individual queries in $O(\log\log n)$ time using the techniques presented below.

A key component of our result is a solution to the following simplified problem. A co-occurrence $[i, j]$ is *left-minimal* if $[i + 1, j]$ is not a co-occurrence. The *left-minimal co-occurrence problem* is to preprocess S and Q into a data structure that supports the query

– $\mathsf{lmco}_{S,Q}(w)$: return the number of left-minimal co-occurrences of Q in S that have length w.

We first solve this more restricted problem, and then we solve the co-occurrence problem by a reduction to the left-minimal co-occurrence problem. To our knowledge this problem has not been studied before.

1.1 Our Results

Our two main contributions are as follows. Firstly, we give an upper bound that matches and simplifies the state of the art. Secondly, we provide lower bounds that show that our solution has optimal space, and that our query time is optimal for optimal-space data structures. As in previous work, all our results work on the word RAM model with logarithmic word size.

To do so we use the following parametrization. Let $\delta_{S,Q}$ be the difference encoding of the sequence $\mathsf{lmco}_{S,Q}(1), \ldots, \mathsf{lmco}_{S,Q}(n)$. That is, $\delta_{S,Q}(i) = \mathsf{lmco}_{S,Q}(i) - \mathsf{lmco}_{S,Q}(i-1)$ for each $i \in [2,n]$ (note that $\mathsf{lmco}(1) = 0$ since $|Q| \geq 2$). Let $Z_{S,Q} = \{i \in [2,n] \mid \delta(i) \neq 0\}$ and let $d_{S,Q} = |Z_{S,Q}|$. For the remainder of the paper we will omit the subscript on lmco, co, Z, and d whenever S and Q are clear from the context. Note that d is a parameter of the problem since it is determined exclusively by the input S and Q. We prove the following theorem.

Theorem 1. *Let S be a string of length n over an alphabet Σ, let Q be a subset of Σ of size $q \geq 2$, and let d be defined as above.*

(a) There is an $O(d)$ space data structure that supports both $\mathsf{lmco}_{S,Q}$- and $\mathsf{co}_{S,Q}$-queries in $O(\log\log n)$ time. The preprocessing algorithm does a single pass over S, runs in expected $O(n)$ time and uses $O(d+q)$ space in addition to the input.

(b) Any data structure supporting either $\mathsf{lmco}_{S,Q}$- or $\mathsf{co}_{S,Q}$-queries needs $\Omega(d)$ space in the worst case, and any $d\log^{O(1)} d$ space data structure cannot support queries faster than $\Omega(\log\log n)$ time.

(c) The parameter d is bounded by $O(\sqrt{nq})$, and any data structure supporting either $\mathsf{lmco}_{S,Q}$- or $\mathsf{co}_{S,Q}$-queries needs $\Omega(\sqrt{nq})$ bits *of space in the worst case.*

Theorem 1(a) and 1(b) together prove that our data structure has optimal space, and that with optimal space we cannot hope to support queries faster than $O(\log\log n)$ time. In comparison to the state of the art by Sobel et al. [14], Theorem 1(c) proves that we match their $O(\sqrt{nq})$ space and $O(\log\log n)$ time solution, and also that the $O(\sqrt{nq})$ space bound is tight to within polylogarithmic factors. All of our results are based on simple and intuitive combinatorial ideas that simplify the state of the art.

Given a set X of m integers from a universe U, the *static predecessor problem* is to represent X such that we can efficiently answer the query $\mathsf{predecessor}(x) = \max\{y \in X \mid y \leq x\}$. Tight bounds by Pătraşcu and Thorup [13] imply that

$O(\log\log|U|)$-time queries are optimal with $m\log^{O(1)} m$ space when $|U| = m^c$ for any constant $c > 1$. The lower bound on query time in Theorem 1(b) follows from the following theorem, which in turn follows from a reduction from the predecessor problem to the (left-minimal) co-occurrence problem.

Theorem 2. *Let $X \subseteq \{2,\dots,u\}$ for some u and let $|X| = m$. Let n, q, and d be the parameters of the (left-minimal) co-occurrence problem as above. Given a data structure that supports lmco- or co-queries in $f_t(n,q,d)$ time using $f_s(n,q,d)$ space, we obtain a data structure that supports predecessor queries on X in $O(f_t(2u^2,2,8m))$ time using $O(f_s(2u^2,2,8m))$ space.*

In particular, if $f_s(n,q,d) = d\log^{O(1)} d$ then we obtain an $m\log^{O(1)} m$-space predecessor data structure on X. If also $u = m^c$, then it follows from the lower bound on predecessor queries that $f_t(2u^2,2,8m) = \Omega(\log\log u)$, which in turn implies that $f_t(n,q,d) = \Omega(\log\log n)$, proving the lower bound in Theorem 1(b).

The preprocessing algorithm and the proof of Theorem 2 can be found in Appendices A and B, respectively.

1.2 Techniques

The key technical insights that lead to our results stem mainly from the structure of δ.

To achieve the upper bound for lmco-queries we use the following very simple data structure. By definition, $\mathsf{lmco}(w) = \sum_{i=2}^{w} \delta(i)$. Furthermore, by the definition of Z it follows that for any $w \in [2,n]$ we have that $\mathsf{lmco}(w) = \mathsf{lmco}(w_p)$ where w_p is the predecessor of w in Z. Our data structure is a predecessor structure over the set of key-value pairs $\{(i,\mathsf{lmco}(i)) \mid i \in Z\}$ and answers lmco-queries with a single predecessor query. There are linear space predecessor structures that support queries in $O(\log\log|U|)$ time [15]. Here the universe U is $[2,n]$ so we match the $O(d)$ space and $O(\log\log n)$ time bound in Theorem 1(a).

Furthermore, we prove the $O(\sqrt{nq})$ upper bound on space by bounding $d = O(\sqrt{nq})$ using the following idea. In essence, each $\delta(z)$ for $z \in Z$ corresponds to some length-z *minimal co-occurrence*, which is a co-occurrence $[i,j]$ such that neither $[i+1,j]$ nor $[i,j-1]$ are co-occurrences (see below for the full details on δ). We bound the cumulative length of all the minimal co-occurrences to be $O(nq)$; then there are at most $d = O(\sqrt{nq})$ distinct lengths of minimal co-occurrences since $d = \omega(\sqrt{nq})$ implies that the cumulative length of the minimal co-occurrences is at least $1 + \ldots + d = \Omega(d^2) = \omega(nq)$.

To also support co-queries and complete the upper bound we give a straightforward reduction from the co-occurrence problem to the left-minimal co-occurrence problem. We show that by extending the above data structure to also store $\sum_{i=2}^{z} \mathsf{lmco}(i)$ for each $z \in Z$, we can support co-queries with the same bounds as for lmco-queries.

On the lower bounds side, we give all the lower bounds for the left-minimal co-occurrence problem and show that they extend to the co-occurrence problem. To prove the lower bounds we exploit that we can carefully design lmco-instances that result in a particular difference encoding δ by including minimal co-occurrences of certain lengths and spacing. Our lower bounds on space in

Theorem 1(b) and 1(c) are the results of encoding a given permutation or set in δ, respectively.

Finally, as mentioned above, we prove Theorem 2 (and, by extension, the lower bound on query time in Theorem 1(b)) by encoding a given instance of the static predecessor problem in an lmco-instance such that the predecessor of an element x equals $\mathsf{lmco}(x)$.

2 The Left-Minimal Co-occurrence Problem

2.1 Main Idea

Let $[i, j] \subseteq [1, n]$ be a co-occurrence of Q in S. Recall that then each character from Q occurs in $S[i, j]$ and that $[i, j]$ is left-minimal if $[i + 1, j]$ is not a co-occurrence. The goal is to preprocess S and Q to support the query $\mathsf{lmco}(w)$, which returns the number of left-minimal co-occurrences of length w.

We say that $[i, j]$ is a *minimal co-occurrence* if neither $[i + 1, j]$ nor $[i, j - 1]$ are co-occurrences and we denote the μ minimal co-occurrences of Q in S by $[\ell_1, r_1], \ldots, [\ell_\mu, r_\mu]$ where $r_1 < \ldots < r_\mu$. This ordering is unique since at most one minimal co-occurrence ends at a given index. To simplify the presentation we define $r_{\mu+1} = n + 1$. Note that also $\ell_1 < \ldots < \ell_\mu$ due to the following property.

Property 1. Let $[a, b]$ and $[a', b']$ be two minimal co-occurrences. Either both $a < a'$ and $b < b'$, or both $a' < a$ and $b' < b$.

Proof. If $a < a'$ and $b \geq b'$ then $[a, b]$ strictly contains another minimal co-occurrence $[a', b']$ and can therefore not be minimal itself. The other cases are analogous. □

We now show that given all the minimal co-occurrences we can determine all the left-minimal co-occurrences.

Lemma 1. *Let $[\ell_1, r_1], \ldots, [\ell_\mu, r_\mu]$ be the minimal co-occurrences of Q in S where $r_1 < \ldots < r_\mu$ and let $r_{\mu+1} = n + 1$. Then*

(a) there are no left-minimal co-occurrences that end before r_1, i.e., at an index $k < r_1$, and
(b) for each index k where $r_i \leq k < r_{i+1}$ for some i, the left-minimal co-occurrence ending at k starts at ℓ_i.

Proof. See Fig. 1 for an illustration of the proof.

(a) If $[j, k]$ is a left-minimal co-occurrence where $k < r_1$, it must contain some minimal co-occurrence that ends before r_1—obtainable by shrinking $[j, k]$ maximally—leading to a contradiction.
(b) Let $r_i \leq k < r_{i+1}$. Then $[\ell_i, k]$ is a co-occurrence since it contains $[\ell_i, r_i]$. We show that it is left-minimal by showing that $[\ell_i + 1, k]$ is not a co-occurrence. If it were, it would contain a minimal co-occurrence $[s, t]$ where $\ell_i < s$ and $t < r_{i+1}$. By Property 1, $\ell_i < s$ implies that $r_i < t$. However, then $r_i < t < r_{i+1}$, leading to a contradiction. □

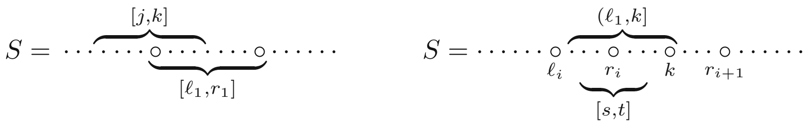


Fig. 1. *Left (Lemma 1(a)):* Any left-minimal co-occurrence $[j, k]$ must contain a minimal co-occurrence ending at or before k. If $k < r_1$ this contradicts that r_1 is the smallest endpoint of a minimal co-occurrence. *Right (Lemma 1(b)):* $[\ell_i, k]$ is a co-occurrence because it contains $[\ell_i, r_i]$. However, $[\ell_i + 1, k]$ is not a co-occurrence; if it were it would contain a minimal co-occurrence $[s, t]$ that ends between r_i and k, leading to a contradiction since $r_i < t < r_{i+1}$.

Let $\mathsf{len}(i, j) = j - i + 1$ denote the length of the interval $[i, j]$. Lemma 1 implies that each minimal co-occurrence $[\ell_i, r_i]$ gives rise to one additional left-minimal co-occurrence of length k for $k = \mathsf{len}(\ell_i, r_i), \ldots, \mathsf{len}(\ell_i, r_{i+1}) - 1$. Also, note that each left-minimal co-occurrence is determined by a minimal co-occurrence in this manner. Therefore, $\mathsf{lmco}(w)$ equals the number of minimal co-occurrences $[\ell_i, r_i]$ where $\mathsf{len}(\ell_i, r_i) \leq w < \mathsf{len}(\ell_i, r_{i+1})$. Recall that $\delta(i) = \mathsf{lmco}(i) - \mathsf{lmco}(i-1)$ for $i \in [2, n]$. Since $\mathsf{lmco}(1) = 0$ (because $|Q| \geq 2$) we have $\mathsf{lmco}(w) = \sum_{i=2}^{w} \delta(i)$. It follows that

$$\delta(w) = \sum_{i=1}^{\mu} \begin{cases} 1 & \text{if } \mathsf{len}(\ell_i, r_i) = w \\ -1 & \text{if } \mathsf{len}(\ell_i, r_{i+1}) = w \\ 0 & \text{otherwise} \end{cases}$$

since the contribution of each $[\ell_i, r_i]$ to the sum $\sum_{i=2}^{w} \delta(i)$ is one if $\mathsf{len}(\ell_i, r_i) \leq w < \mathsf{len}(\ell_i, r_{i+1})$ and zero otherwise. We say that $[\ell_i, r_i]$ *contributes* plus one and minus one to $\delta(\mathsf{len}(\ell_i, r_i))$ and $\delta(\mathsf{len}(\ell_i, r_{i+1}))$, respectively.

However, note that only the non-zero $\delta(\cdot)$-entries affect the result of lmco-queries. Denote the elements of Z by $z_1 < z_2 < \ldots < z_d$ and define $\mathsf{pred}(w)$ such that $z_{\mathsf{pred}(w)}$ is the predecessor of w in Z, or $\mathsf{pred}(w) = 0$ if w has no predecessor. We get the following lemma.

Lemma 2. *For any $w \in [z_1, n]$ we have that*

$$\mathsf{lmco}(w) = \sum_{i=1}^{\mathsf{pred}(w)} \delta(z_i) = \mathsf{lmco}(z_{\mathsf{pred}(w)}).$$

For $w \in [0, z_1)$, w has no predecessor in Z and $\mathsf{lmco}(w) = 0$.

Proof. The proof follows from the fact that $\mathsf{lmco}(w) = \sum_{i=2}^{w} \delta(i)$ and $\delta(i) = 0$ for each $i \notin Z$. $\square$

2.2 Data Structure

The contents of the data structure are as follows. Store the linear space predecessor structure from [15] over the set Z, and for each key $z_i \in Z$ store the data $\mathsf{lmco}(z_i)$. To answer $\mathsf{lmco}(w)$, find the predecessor $z_{\mathsf{pred}(w)}$ of w and return $\mathsf{lmco}(z_{\mathsf{pred}(w)})$. Return 0 if w has no predecessor.

The correctness of the query follows from Lemma 2. The query time is $O(\log\log|U|)$ [15] which is $O(\log\log n)$ since the universe is $[2, n]$, i.e., the domain of δ. The predecessor structure uses $O(|Z|) = O(d)$ space, which we now show is $O(\sqrt{nq})$. We begin by bounding the cumulative length of the minimal co-occurrences.

Lemma 3. *Let* $[\ell_1, r_1], \ldots, [\ell_\mu, r_\mu]$ *be the minimal co-occurrences of* Q *in* S. *Then*

$$\sum_{i=1}^{\mu} \mathit{len}(\ell_i, r_i) = O(nq).$$

Proof. We prove that for each $k \in [1, n]$ there are at most q minimal co-occurrences $[\ell_i, r_i]$ where $k \in [\ell_i, r_i]$; the statement in the lemma follows directly. Suppose that there are $q' > q$ minimal co-occurrences $[s_1, t_1], \ldots, [s_{q'}, t_{q'}]$ that contain k and let $t_1 < \ldots < t_{q'}$. By Property 1, and because each minimal occurrence contains k, we have

$$s_1 < \ldots < s_{q'} \leq \ k \ \leq t_1 < \ldots < t_{q'}$$

Furthermore, for each s_i we have that $S[s_i] = p$ for some $p \in Q$; otherwise $[s_i + 1, t_i]$ would be a co-occurrence and $[s_i, t_i]$ would not be minimal. Since $q' > q$ there is some $p \in Q$ that occurs twice as the first character, i.e., such that $S[s_i] = S[s_j] = p$ for some $i < j$. However, then $[s_i + 1, t_i]$ is a co-occurrence because it still contains $S[s_j] = p$, contradicting that $[s_i, t_i]$ is minimal. □

By the definition of δ, we have that $\delta(k) \neq 0$ only if there is some minimal co-occurrence $[\ell_i, r_i]$ such that either $\mathsf{len}(\ell_i, r_i) = k$ or $\mathsf{len}(\ell_i, r_{i+1}) = k$. Using this fact in conjunction with Lemma 3 we bound the sum of the elements in Z.

$$\begin{aligned}
\sum_{z \in Z} z = \sum_{k \text{ where } \delta(k) \neq 0} k \quad &\leq \sum_{i=1}^{\mu} \mathsf{len}(\ell_i, r_i) + \mathsf{len}(\ell_i, r_{i+1}) \\
&= \sum_{i=1}^{\mu} \mathsf{len}(\ell_i, r_i) + \Big(\mathsf{len}(\ell_i, r_i) + \mathsf{len}(r_i + 1, r_{i+1})\Big) \\
&= \sum_{i=1}^{\mu} 2 \cdot \mathsf{len}(\ell_i, r_i) + \sum_{i=1}^{\mu} \mathsf{len}(r_i + 1, r_{i+1}) \\
&= O(nq) + O(n)
\end{aligned}$$

Since the sum over Z is at most $O(nq)$ we must have $d = O(\sqrt{nq})$, because with $d = \omega(\sqrt{nq})$ distinct elements in Z we have

$$\sum_{z \in Z} z \geq 1 + 2 + \ldots + d = \Omega(d^2) = \omega(nq).$$

3 The Co-occurrence Problem

Recall that $\mathsf{co}(w)$ is the number of co-occurrences of length w, as opposed to the number of left-minimal co-occurrences of length w. That is, $\mathsf{co}(w)$ counts the number of co-occurrences among the intervals $[1, w], [2, w+1], \ldots, [n-w+1, n]$. We reduce the co-occurrence problem to the left-minimal co-occurrence problem as follows.

Lemma 4. *Let S be a string over an alphabet Σ, let $Q \subseteq \Sigma$ and let lmco be defined as above. Then*

$$\mathsf{co}(w) = \left(\sum_{i=2}^{w} \mathsf{lmco}(i)\right) - \max(w - r_1, 0).$$

Proof. For any index $k \geq w$, the length-w interval $[k-w+1, k]$ ending at k is a co-occurrence if and only if the length of the left-minimal co-occurrence ending at index k is at most w. The sum $\sum_{i=2}^{w} \mathsf{lmco}(i)$ counts the number of indices $j \in [1, n]$ such that the left-minimal co-occurrence ending at j has length at most w. However, this also includes the left-minimal co-occurrences that end at any index $j \in [r_1, w-1]$. While all of these have length at most $w-1$, none of the length-w intervals that end in the range $[r_1, w-1]$ correspond to substrings of S, so they are not co-occurrences. Therefore, the sum $\sum_{i=2}^{w} \mathsf{lmco}(i)$ overestimates $\mathsf{co}(w)$ by $w - r_1$ if $r_1 < w$ and by 0 otherwise. □

We show how to represent the sequence $\sum_{i=2}^{2} \mathsf{lmco}(i), \ldots, \sum_{i=2}^{n} \mathsf{lmco}(i)$ compactly, in a similar way to what we did for lmco-queries. Recall that the elements of Z are denoted by $z_1 < \ldots < z_d$, that $z_{\mathsf{pred}(x)}$ is the predecessor of x in Z, and that $\mathsf{pred}(x) = 0$ if x has no predecessor. Then, for any $w \geq 2$ we get that

$$\begin{aligned}
\sum_{i=2}^{w} \mathsf{lmco}(i) &= \sum_{i=2}^{w} \sum_{j=1}^{\mathsf{pred}(i)} \delta(z_j) \\
&= \sum_{k=1}^{\mathsf{pred}(w)} \delta(z_k)(w - z_k + 1) \\
&= (w+1) \underbrace{\sum_{k=1}^{\mathsf{pred}(w)} \delta(z_k)}_{\mathsf{lmco}(w)} - \sum_{k=1}^{\mathsf{pred}(w)} z_k \delta(z_k)
\end{aligned} \tag{1}$$

The first step follows by Lemma 2 and the second step follows because $\delta(z_k)$ occurs in $\sum_{j=1}^{\mathsf{pred}(i)} \delta(z_j)$ for each of the $w - z_k + 1$ choices of $i \in [z_k, w]$.

To also support co-queries we extend our data structure from before as follows. For each z_k in the predecessor structure we store $\sum_{i=1}^{k} z_i \delta(z_i)$ in addition to $\mathsf{lmco}(z_k)$. We also store r_1. Using Lemma 4 and Eq. 1 we can then answer co-queries with a single predecessor query and a constant amount of extra work, taking $O(\log \log n)$ time. The space remains $O(d) = O(\sqrt{nq})$. This completes the proof of Theorem 1(a), as well as the upper bound on space from Theorem 1(c).

4 Lower Bounds

In this section we show lower bounds on the space complexity of data structures that support lmco- or co-queries. In Sect. 4.1 we introduce a gadget that we use in Sect. 4.2 to prove that any data structure supporting lmco- or co-queries needs $\Omega(d)$ space (we use the same gadget in Appendix B to prove Theorem 2). In Sect. 4.3 we prove that any solution to the (left-minimal) co-occurrence problem requires $\Omega(\sqrt{nq})$ bits of space in the worst case.

All the lower bounds are proven by reduction to the left-minimal co-occurrence problem. However, they extend to data structures that support co-queries by the following argument. Store r_1 and any data structure that supports co on S and Q in time t per query. Then this data structure supports lmco-queries in $O(t)$ time, because by Lemma 4 we have that

$$\begin{aligned}
&\mathsf{co}(w) - \mathsf{co}(w-1)\\
&\quad= \left(\sum_{i=2}^{w} \mathsf{lmco}(i) - \max(w-r_1,0)\right) - \left(\sum_{i=2}^{w-1} \mathsf{lmco}(i) - \max(w-1-r_1,0)\right)\\
&\quad= \mathsf{lmco}(w) - \max(w-r_1,0) + \max(w-1-r_1,0)
\end{aligned}$$

4.1 The Increment Gadget

Let $Q = \{\mathtt{A},\mathtt{B}\}$ and $U = \{2,\ldots,u\}$. For each $i \in U$ we define the *increment gadget*

$$G_i = \underbrace{\mathtt{A}\,\$\cdots\$\,\mathtt{B}}_{i}\ \underbrace{\$\cdots\$}_{u}$$

where $\$\cdots\$$ denotes a sequence of characters that are not in Q.

Lemma 5. *Let $Q = \{\mathtt{A},\mathtt{B}\}$, $U = \{2,\ldots,u\}$, and let G_i be defined as above. Furthermore, for some $E = \{e_1, e_2, \ldots, e_m\} \subseteq U$ let S be the concatenation of $c_1 > 0$ copies of G_{e_1}, with $c_2 > 0$ copies of G_{e_2}, and so on. That is,*

$$S = \underbrace{G_{e_1}\cdots G_{e_1}}_{c_1}\ \cdots\cdots\cdots\ \underbrace{G_{e_m}\cdots G_{e_m}}_{c_m}$$

Then $\delta(e_i) = c_i$ for each $e_i \in E$ and $\delta(e) = 0$ for any $e \in U \setminus E$. Furthermore, $m \le d \le 8m$ and $n \le 2uC$ where $C = \sum_{i=1}^{m} c_i$ is the number of gadgets in S.

Proof. Firstly, $|G_j| = j + u \le 2u$ since $j \in U$, so the combined length of the C gadgets is at most $2uC$.

Now we prove that $\delta(e_i) = c_i$ for each $e_i \in E$ and $\delta(e) = 0$ for each $e \in U \setminus E$. Consider two gadgets G_j and G_k that occur next to each other in S.

$$\overbrace{\underbrace{\mathtt{A}\,\$\cdots\$\,\mathtt{B}}_{j}\ \underbrace{\$\cdots\$}_{u}}^{G_j}\ \overbrace{\underbrace{\mathtt{A}\,\$\cdots\$\,\mathtt{B}}_{k}\ \underbrace{\$\cdots\$}_{u}}^{G_k}$$

Three of the minimal co-occurrences in S occur in these two gadgets. Denote them by $[s_1, t_1]$,$[s_2, t_2]$ and $[s_3, t_3]$.

$$\underbrace{\texttt{A}\,\$\cdots\$\,\texttt{B}}_{[s_1,t_1]}\!\!\!\overbrace{\phantom{\texttt{B}}\,\$\cdots\$\,\phantom{\texttt{A}}}^{[s_2,t_2]}\!\!\!\underbrace{\texttt{A}\,\$\cdots\$\,\texttt{B}}_{[s_3,t_3]}\ \$\cdots\$$$

The two first minimal co-occurrences start in G_j. They contribute

- plus one to $\delta(x)$ for $x \in \{\mathsf{len}(s_1, t_1), \mathsf{len}(s_2, t_2)\} = \{j, u + 2\}$.
- minus one to $\delta(x)$ for $x \in \{\mathsf{len}(s_1, t_2), \mathsf{len}(s_2, t_3)\} = \{j + u + 1, k + u + 1\}$.

Hence, each occurrence of G_j contributes plus one to $\delta(j)$, and the remaining contributions are to $\delta(x)$ where $x \notin U$. The argument is similar also for the last gadget in S that has no other gadget following it. For each $e_i \in E$ there are c_i occurrences of G_{e_i} so $\delta(e_i) = c_i$. For each $e \in U \setminus E$ there are no occurrences of G_e so $\delta(e) = 0$.

Finally, note that each occurrence of $G_j G_k$ at different positions in S contributes to the same four $\delta(\cdot)$-entries. Therefore the number of distinct non-zero $\delta(\cdot)$-entries is linear in the number of distinct pairs (j, k) such that G_j and G_k occur next to each other in S. Here we have no more than $2m$ distinct paris since G_{e_i} is followed either by G_{e_i} or $G_{e_{i+1}}$. Each distinct pair contributes to at most four $\delta(\cdot)$-entries so $d \le 8$ m. Finally each G_{e_i} contributes at least to $\delta(e_i)$ so $m \le d$, concluding the proof. $\square$

4.2 Lower Bound on Space

We prove that any data structure supporting lmco-queries needs $\Omega(d)$ space in the worst case. Let U and Q be defined as in the increment gadget and let $P = p_2, \ldots, p_m$ be a sequence of length $m-1$ where each $p_i \in U$ (the first element is named p_2 for simplicity). We let S be the concatenation of p_2 occurrences of G_2, with p_3 occurrences of G_3, and so on. That is,

$$S = \underbrace{G_2 \ldots G_2}_{p_2} \cdots\cdots\cdots \underbrace{G_m \ldots G_m}_{p_m}$$

Then any data structure supporting lmco on S and Q is a representation of P; by Lemma 5 we have that $\delta(i) = p_i$ for $i \in [2, m]$ and by definition we have $\delta(i) = \mathsf{lmco}(i) - \mathsf{lmco}(i-1)$.

The sequence P can be any one of $(u-1)^{m-1}$ distinct sequences, so any representation of P requires

$$\log((u-1)^{m-1}) = (m-1)\log(u-1) = \Omega(m \log u)$$

bits—or $\Omega(m)$ words—in the worst case. By Lemma 5 this is $\Omega(d)$.

4.3 Lower Bound on Space in Terms of n and q

Here we prove that any data structure supporting lmco needs $\Omega(\sqrt{nq})$ bits of space in the worst case.

The main idea is as follows. Given an integer α and some $k \in \{2, \ldots, \alpha\}$, let V be the set of *even* integers from $\{k+1, \ldots, k\alpha\}$, and let T be some subset of V. We will construct an instance S and Q where

- the size of Q is $q = k$
- the length of S is $n = O(k\alpha^2)$
- for each $i \in V$ we have $\delta(i) = 1$ if and only if $i \in T$.

Then, as above, any data structure supporting lmco-queries on S and Q is a representation of T since $\delta(i) = \mathsf{lmco}(i) - \mathsf{lmco}(i-1)$. There are $2^{\Omega(k\alpha)}$ choices for T, so any representation of T requires

$$\log 2^{\Omega(k\alpha)} = \Omega(k\alpha) = \Omega(\sqrt{k^2\alpha^2}) = \Omega(\sqrt{nq})$$

bits in the worst case.

The reduction is as follows. Let $Q = \{\mathtt{C}_1, \ldots, \mathtt{C}_k\}$ and let $\mathtt{\$}$ be a character not in Q. Assume for now that $|T|$ is a multiple of $k-1$ and partition T arbitrarily into $t = O(\alpha)$ sets $T_1, \ldots, T_t$, each of size $k-1$. Consider $T_j = \{e_1, \ldots, e_{k-1}\}$ where $e_1 < e_2 < \ldots < e_{k-1}$. We encode T_j in the gadget R_j where

- the length of R_j is $3k\alpha$.
- $R_j[1,k] = \mathtt{C}_1\mathtt{C}_2 \ldots \mathtt{C}_k$.
- $R_j[i + e_i] = \mathtt{C}_i$ for each $\mathtt{C}_i$ except $\mathtt{C}_k$. This is always possible since $i + e_i < (i+1) + e_{i+1}$.
- all other characters are $\mathtt{\$}$.

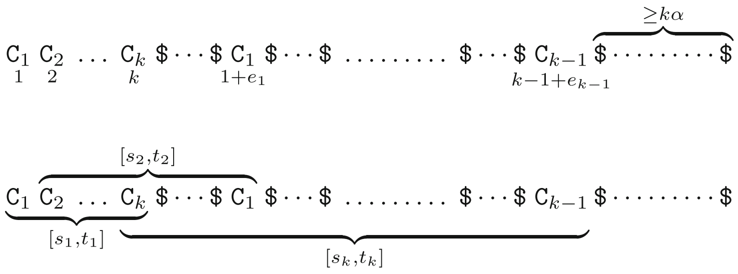
See Fig. 2 for an illustration both of the layout of R_j and of the minimal co-occurrences contained within it. There are k minimal co-occurrences contained in R_j which we denote by $[s_1, t_1], \ldots, [s_k, t_k]$.

- The first one, $[s_1, t_1] = [1, k]$, starts and ends at the first occurrence of $\mathtt{C}_1$ and $\mathtt{C}_k$, respectively.
- For each $i \in [2, k]$, the minimal co-occurrence $[s_i, t_i]$ starts at the first occurrence of $\mathtt{C}_i$ and ends at the second occurrence of $\mathtt{C}_{i-1}$, i.e., $[s_i, t_i] = [i, i-1+e_{i-1}]$.

Consider how these minimal co-occurrences contribute to δ. Each $[s_i, t_i]$ contributes plus one to $\delta(\mathsf{len}(s_i, t_i))$. For the first co-occurrence, $\mathsf{len}(s_1, t_1) = k$ (which is not a part of the universe V). Each of the other co-occurrences $[s_i, t_i]$ has length

$$\mathsf{len}(s_i, t_i) \;=\; t_i - s_i + 1 \;=\; (i - 1 + e_{i-1}) - i + 1 \;=\; e_{i-1}$$

Therefore, the remaining minimal co-occurrences contribute plus one to of each the $\delta(\cdot)$-entries $e_1, \ldots, e_{k-1}$.



$\geq k\alpha$
$\mathtt{C}_1\ \mathtt{C}_2\ \ldots\ \mathtt{C}_k\ \$\cdots\$\ \mathtt{C}_1\ \$\cdots\$\ \ldots\ldots\ldots\ \$\cdots\$\ \mathtt{C}_{k-1}\ \$\cdots\cdots\cdots\$$
1
2
k
$1{+}e_1$
$k{-}1{+}e_{k-1}$
$[s_2,t_2]$
$[s_1,t_1]$
$[s_k,t_k]$

Fig. 2. *Top:* Shows the layout of the gadget R_j, where $\$\cdots\$$ denotes a sequence of characters not in Q. The first k characters are $\mathtt{C}_1 \ldots \mathtt{C}_k$. For $i \in [1, k-1]$ there is another occurrence of $\mathtt{C}_i$ at index $i + e_i$. Note that the second occurrence of $\mathtt{C}_1$ occurs before the second occurrence of $\mathtt{C}_2$, and so on. All other characters are $\$$. Since $|R_j| = 3k\alpha$ and $k-1+e_{k-1} \leq 2k\alpha$, R_j ends with at least $k\alpha$ characters that are not in Q. *Bottom:* Shows the k minimal co-occurrences in R_j denoted by $[s_1, t_1], \ldots, [s_k, t_k]$. Each of the $k-3$ minimal co-occurrences that are not depicted start at the first occurrence of some $\mathtt{C}_i$ and ends at the second occurrence of $\mathtt{C}_{i-1}$.

Furthermore, each $[s_i, t_i]$ contributes negative one to $\delta(\mathsf{len}(s_i, t_{i+1}))$, where we define $t_{k+1} = |R_j| + 1$. For $i < k$ we have that $\mathsf{len}(s_i, t_{i+1}) = 1 + \mathsf{len}(s_{i+1}, t_{i+1}) = 1 + e_i$ since $s_i + 1 = s_{i+1}$. Note that $1 + e_i$ is odd since e_i is even, and therefore not in V. For $i = k$, we get that $t_{i+1} = t_{k+1} = |R_j| + 1$. The last $k\alpha$ (at least) characters of R_j are not in Q, so $\mathsf{len}(s_k, |R_j| + 1) > k\alpha$ and therefore not in V.

Hence, R_j contributes plus one to $\delta(e_1), \ldots, \delta(e_{k-1})$ and does not contribute anything to $\delta(i)$ for any other $i \in V \backslash T_j$. To construct S, concatenate $R_1, \ldots, R_t$. Note that any minimal co-occurrence that crosses the boundary between two gadgets will only contribute to $\delta(i)$ for $i > k\alpha$ due to the trailing characters of each gadget that are not in Q. Since S consists of $t = O(\alpha)$ gadgets that each have length $O(k\alpha)$, we have $n = O(k\alpha^2)$ as stated above.

Finally, note that the assumption that $|T|$ is a multiple of $k-1$ is not necessary. We ensure that the size is a multiple of $k-1$ by adding at most $k-2$ *even* integers from $\{k\alpha + 1, \ldots, 2k\alpha\}$ and adjusting the size of the gadgets accordingly. The reduction still works because we add even integers, the size of S is asymptotically unchanged, and any minimal co-occurrence due to the extra elements will have length greater than $k\alpha$ and will not contribute to any relevant δ-entries.

Acknowledgement. We would like to thank the anonymous reviewers for their comments, which improved the presentation of the paper.

A Preprocessing

Finding Minimal Co-occurrences. To build the data structure, we need to find all the minimal co-occurrences in order to determine δ. For $j \geq r_1$, let $\mathsf{lm}(j)$ denote the length of the left-minimal co-occurrence ending at index j. By Lemma 1,

$\mathsf{lm}(r_i) = \mathsf{len}(\ell_i, r_i)$ for each $i \in [1, \mu]$. Furthermore, for $j \in [r_i + 1, r_{i+1} - 1]$ we have $\mathsf{lm}(j) = \mathsf{lm}(j-1) + 1$ since both of the left-minimal co-occurrences ending at these two indices start at ℓ_i. However, $\mathsf{lm}(r_i) \leq \mathsf{lm}(r_i - 1)$ for each $i \in [2, \mu]$; the left-minimal co-occurrence ending at r_i starts at least one index further to the right than the left-minimal co-occurrence ending at $r_i - 1$ because $\ell_{i-1} < \ell_i$, so it cannot be strictly longer.

We determine $\ell_1, \ldots, \ell_\mu$ and $r_1, \ldots, r_\mu$ using the following algorithm. Traverse S and maintain $\mathsf{lm}(j)$ for the current index j. Whenever $\mathsf{lm}(j) \neq \mathsf{lm}(j-1)+1$ the interval $[j - \mathsf{lm}(j) + 1,\ j]$ is one of the minimal co-occurrences. Note that this algorithm finds the minimal co-occurrences in order by their rightmost endpoint. We maintain $\mathsf{lm}(j)$ as follows. For each character $p \in Q$ let $\mathsf{dist}(j, p)$ be the distance to the closest occurrence of p on the left of j. Then $\mathsf{lm}(j)$ is the maximum $\mathsf{dist}(j, \cdot)$-value. As in [14], we maintain the $\mathsf{dist}(j, \cdot)$-values in a linked list that is dynamically reordered according to the well-known *move-to-front* rule. The algorithm works as follows. Maintain a linked list over the elements in Q, ordered by increasing dist-values. Whenever you see some $p \in Q$, access its node in expected constant time through a dictionary and move it to the front of the list. The least recently seen $p \in Q$ (i.e., the p with the largest $\mathsf{dist}(j, \cdot)$-value) is found at the back of the list in constant time. The algorithm uses $O(q)$ space and expected constant time per character in S, thus it runs in expected $O(n)$ time.

Building the Data Structure We build the data structure as follows. Traverse S and maintain the two most recently seen minimal co-occurrences using the algorithm above. We maintain the non-zero $\delta(\cdot)$-values in a dictionary D that is implemented using chained hashing in conjunction with universal hashing [2]. When we find a new minimal co-occurrence $[\ell_{i+1}, r_{i+1}]$ we increment $D[\mathsf{len}(\ell_i, r_i)]$ and decrement $D[\mathsf{len}(\ell_i, r_{i+1})]$. Recall that $Z = \{z_1, \ldots, z_d\}$ where $z_j < z_{j+1}$ is defined such that $\delta(i) \neq 0$ if and only if $i \in Z$. After processing S the dictionary D encodes the set $\{(z_1, \delta(z_1)), \ldots, (z_d, \delta(z_d))\}$. Sort the set to obtain the array $E[j] = \delta(z_j)$. Compute the partial sum array over E, i.e. the array

$$F[j] = \sum_{i=1}^{j} E[i] = \sum_{i=1}^{j} \delta(z_i) = \mathsf{lmco}(z_j). \qquad \text{(we use 1-indexing)}$$

Build the predecessor data structure over Z and associate $\mathsf{lmco}(z_j)$ with each key z_j.

The algorithm for finding the minimal co-occurrences uses $O(q)$ space and the remaining data structures all use $O(d)$ space, for a total of $O(d + q)$ space. Finding the minimal co-occurrences and maintaining D takes $O(n)$ expected time, and so does building the predecessor structure from the sorted input.

Furthermore, we use the following sorting algorithm to sort the d entries in D with $O(d)$ extra space in expected $O(n)$ time. If $d < n/\log n$ we use merge sort which uses $O(d)$ extra space and runs in $O(d \log d) = O(n)$ time. If $d \geq n/\log n$ we use radix sort with base $\sqrt{n}$, which uses $O(\sqrt{n})$ extra space and $O(n)$ time. To elaborate, assume without loss of generality that $2k$ bits are necessary to represent n. We first distribute the elements into $2^k = O(\sqrt{n})$ buckets according to the most significant k bits of their binary representation, partially sorting the input. We then sort each bucket by distributing the elements in that bucket according to the *least* significant k bits of their binary representation, fully sorting the input. The algorithm runs in $O(n)$ time and uses $O(\sqrt{n}) = O(n/\log n) = O(d)$ extra space.

B Lower Bound on Time

We now prove Theorem 2 by the following reduction from the predecessor problem. Let U, Q and G_i be as defined in Sect. 4.1 and let $X = \{x_1, x_2, \ldots, x_m\} \subseteq U$ where $x_1 < \ldots < x_m$. Define

$$S = \underbrace{G_{x_1} \cdots G_{x_1}}_{x_1} \underbrace{G_{x_2} \cdots G_{x_2}}_{x_2 - x_1} \cdots\cdots\cdots \underbrace{G_{x_m} \cdots G_{x_m}}_{x_m - x_{m-1}}$$

By Lemma 5 we have that $\delta(x_1) = x_1$, $\delta(x_i) = x_i - x_{i-1}$ for $i \in [2, m]$ and $\delta(i) = 0$ for $i \in U \setminus X$. Then, if the predecessor of some $x \in U$ is x_p, we have

$$\mathsf{lmco}(x) = \sum_{i=2}^{x} \delta(i) = x_1 + (x_2 - x_1) + \ldots + (x_p - x_{p-1}) = x_p$$

On the other hand, if $x < x_1$ then $\sum_{i=0}^{x} \delta(i) = 0$, unambiguously identifying that x has no predecessor.

Applying Lemma 5 again, we have $d \leq 8m$. Furthermore, there are $x_1 + (x_2 - x_1) + \ldots + (x_m - x_{m-1}) = x_m \leq u$ gadgets in total so $n \leq 2u^2$. Hence, given a data structure that supports lmco in $f_t(n, q, d)$ time using $f_s(n, q, d)$ space, we get a data structure supporting predecessor queries on X in $O(f_t(2u^2, 2, 8m))$ time and $O(f_s(2u^2, 2, 8m))$ space, proving Theorem 2.

References

1. Amagata, D., Hara, T.: Mining top-k co-occurrence patterns across multiple streams (extended abstract). In: Proceeding 34th ICDE, pp. 1747–1748 (2018). https://doi.org/10.1109/ICDE.2018.00231
2. Carter, L., Wegman, M.N.: Universal classes of hash functions. J. Comput. Syst. Sci. **18**(2), 143–154 (1979). https://doi.org/10.1016/0022-0000(79)90044-8
3. Chang, J.H., Lee, W.S.: Finding recently frequent item sets adaptively over online transactional data streams. Inf. Syst. **31**(8), 849–869 (2006). https://doi.org/10.1016/j.is.2005.04.001
4. Dallachiesa, M., Palpanas, T.: Identifying streaming frequent items in ad hoc time windows. Data Knowl. Eng. **87**, 66–90 (2013). https://doi.org/10.1016/j.datak.2013.05.007
5. Das, G., Fleischer, R., Gasieniec, L., Gunopulos, D., Kärkkäinen, J.: Episode matching. In: Proceeding 8th CPM, pp. 12–27 (1997). https://doi.org/10.1007/3-540-63220-4_46

6. Demaine, E.D., López-Ortiz, A., Munro, J.I.: Frequency estimation of internet packet streams with limited space. In: Proceeding 10th ESA, pp. 348–360 (2002). https://doi.org/10.1007/3-540-45749-6_33
7. Golab, L., DeHaan, D., Demaine, E.D., López-Ortiz, A., Munro, J.I.: Identifying frequent items in sliding windows over on-line packet streams. In: Proceeding 3rd ACM IMC, pp. 173–178 (2003). https://doi.org/10.1145/948205.948227
8. Karp, R.M., Shenker, S., Papadimitriou, C.H.: A simple algorithm for finding frequent elements in streams and bags. ACM Trans. Database Syst. **28**, 51–55 (2003). https://doi.org/10.1145/762471.762473
9. Li, H., Lee, S.: Mining frequent itemsets over data streams using efficient window sliding techniques. Expert Syst. Appl. **36**(2), 1466–1477 (2009). https://doi.org/10.1016/j.eswa.2007.11.061
10. Lim, Y., Choi, J., Kang, U.: Fast, accurate, and space-efficient tracking of time-weighted frequent items from data streams. In: Proceeding 23rd CIKM, pp. 1109–1118 (2014). https://doi.org/10.1145/2661829.2662006
11. Lin, C., Chiu, D., Wu, Y., Chen, A.L.P.: Mining frequent itemsets from data streams with a time-sensitive sliding window. In: Proceeding 5th SDM, pp. 68–79 (2005). https://doi.org/10.1137/1.9781611972757.7
12. Mozafari, B., Thakkar, H., Zaniolo, C.: Verifying and mining frequent patterns from large windows over data streams. In: Proceeding 24th ICDE, pp. 179–188 (2008). https://doi.org/10.1109/ICDE.2008.4497426
13. Patrascu, M., Thorup, M.: Randomization does not help searching predecessors. In: Proceedimg 18th SODA, pp. 555–564 (2007). https://dl.acm.org/citation.cfm?id=1283383.1283443
14. Sobel, J., Bertram, N., Ding, C., Nargesian, F., Gildea, D.: AWLCO: all-window length co-occurrence. In: Proceeding 32nd CPM, pp. 24:1–24:21. LIPIcs (2021). https://doi.org/10.4230/LIPIcs.CPM.2021.24
15. Willard, D.E.: Log-logarithmic worst-case range queries are possible in space $\Theta(N)$. Inf. Process. Lett. **17**(2), 81–84 (1983). https://doi.org/10.1016/0020-0190(83)90075-3
16. Yu, Z., Yu, X., Liu, Y., Li, W., Pei, J.: Mining frequent co-occurrence patterns across multiple data streams. In: Proceeding 1th EDBT, pp. 73–84 (2015). https://doi.org/10.5441/002/edbt.2015.08