Computing *q*-gram Non-overlapping Frequencies on SLP Compressed Texts

Keisuke Goto, Hideo Bannai, Shunsuke Inenaga, and Masayuki Takeda

Department of Informatics, Kyushu University
744 Motooka, Nishiku, Fukuoka 819–0395, Japan
{`keisuke.gotou,bannai,inenaga,takeda`}`@inf.kyushu-u.ac.jp`

Abstract. Length-q substrings, or q-grams, can represent important characteristics of text data, and determining the frequencies of all q-grams contained in the data is an important problem with many applications in the field of data mining and machine learning. In this paper, we consider the problem of calculating the *non-overlapping frequencies* of all q-grams in a text given in compressed form, namely, as a straight line program (SLP). We show that the problem can be solved in $O(q^2 n)$ time and $O(qn)$ space where n is the size of the SLP. This generalizes and greatly improves previous work (Inenaga & Bannai, 2009) which solved the problem only for $q = 2$ in $O(n^4 \log n)$ time and $O(n^3)$ space.

1 Introduction

In many situations, large-scale text data is first compressed for storage, and then is usually decompressed when it is processed afterwards, where we must again face the size of the data. To circumvent this problem, algorithms that work directly on the compressed representation without explicit decompression have gained attention, especially for the string pattern matching problem [1], and there has been growing interest in what problems can be efficiently solved in this kind of setting [14, 17, 7, 16, 8, 6, 4].

The *non-overlapping occurrence frequency* of a string P in a text string T is defined as the maximum number of non-overlapping occurrences of P in T [3]. Non-overlapping frequencies are required in several grammar based compression algorithms [13, 2], as well as ... In this paper, we consider the problem of computing the non-overlapping occurrence frequencies of *all q*-grams (length-q substrings) occurring in a text T, when the text is given as a *straight line program* (SLP) [10] of size n. An SLP is a context free grammar in the Chomsky normal form that derives a single string. SLPs are a widely accepted abstract model of various text compression schemes, since texts compressed by any grammar-based compression algorithm (e.g. [18, 13]) can be represented as SLPs, and those compressed by the LZ-family (e.g. [19, 20]) can be quickly transformed to SLPs. Theoretically, the length N of the text represented by an SLP of size n can be as large as $O(2^n)$, and therefore a polynomial time algorithm that runs on an SLP representation is, in the worst case, faster than any algorithm which works on the uncompressed string.

For SLP compressed texts, the problem was first considered in [8], where an algorithm for $q = 2$ running in $O(n^4 \log n)$ time and $O(n^3)$ space was presented. However, the algorithm cannot be readily extended to handle $q > 2$. Intuitively, the problem for $q = 2$ is much easier compared to larger values of q, since there is only one way for a 2-gram to overlap, while there can be many ways that a longer q-gram can overlap. In this paper we present the first algorithm for calculating the non-overlapping occurrence frequency of all q-grams, that works for any $q \geq 2$, and runs in $O(q^2 n)$ time and $O(qn)$ space. Not only do we solve a more general problem, but the complexity is greatly improved compared to previous work.

A similar problem for SLPs, where occurrences of q-grams are allowed to overlap, was also considered in [8], where an $O(|\Sigma|^2 n^2)$ time and $O(n^2)$ space algorithm was presented for $q = 2$. A much simpler and efficient $O(qn)$ time and space algorithm for general $q \geq 2$ was recently developed [6]. As is the case with uncompressed strings, ideas from the algorithms allowing overlapping occurrences can be applied *somewhat* to the problem of obtaining non-overlapping occurrence frequencies. However, there are still difficulties that arise from the overlapping of occurrences that must be overcome, i.e., the occurrences of each q-gram can be obtained in the same way, but we must somehow compute their non-overlapping occurrence frequency, which is not a trivial task.

For uncompressed texts, the problem considered in this paper can be solved in $O(|T|)$ time, by applying string indices such as suffix arrays. A similar problem is the *string statistics problem* [3], which asks for the non-overlapping occurrence frequency of a given string P in text string T. The problem can be solved in $O(|P|)$ time for any P, provided that the text is pre-processed in $O(|T| \log |T|)$ time using the sophisticated algorithm of [5]. However, note that the preprocessing requires only $O(|T|)$ time if occurrences are allowed to overlap. This perhaps indicates the intrinsic difficulty that arises when considering overlaps.

2 Preliminaries

2.1 Notation

Let Σ be a finite *alphabet*. An element of Σ^* is called a *string*. The length of a string T is denoted by $|T|$. The empty string ε is a string of length 0, namely, $|\varepsilon| = 0$. A string of length $q > 0$ is called a q-gram. The set of q-grams is denoted by Σ^q. For a string $T = XYZ$, X, Y and Z are called a *prefix*, *substring*, and *suffix* of T, respectively. The i-th character of a string T is denoted by $T[i]$ for $1 \leq i \leq |T|$, and the substring of a string T that begins at position i and ends at position j is denoted by $T[i : j]$ for $1 \leq i \leq j \leq |T|$. For convenience, let $T[i : j] = \varepsilon$ if $j < i$. Let T^R denote the reversal of T, namely, $T^R = T[N]T[N-1] \cdots T[1]$, where $N = |T|$.

For an integer i and a set of integers A, let $i \oplus A = \{i + x \mid x \in A\}$ and $i \ominus A = \{i - x \mid x \in A\}$. If $A = \emptyset$, then let $i \oplus A = i \ominus A = \emptyset$. Similarly, for a pair of integers (x, y), let $i \oplus (x, y) = (i + x, i + y)$.

2.2 Occurrences and Frequencies

For any strings T and P, let $Occ(T, P)$ be the set of occurrences of P in T, i.e.,

$$Occ(T, P) = \{k > 0 \mid T[k : k + |P| - 1] = P\}.$$

The number of occurrences of P in T, or the *frequency* of P in T is, $|Occ(T, P)|$. Any two occurrences $k_1, k_2 \in Occ(T, P)$ with $k_1 < k_2$ are said to be *overlapping* if $k_1 + |P| - 1 \geq k_2$. Otherwise, they are said to be *non-overlapping*. The *non-overlapping frequency* $nOcc(T, P)$ of P in T is defined as the size of a largest subset of $Occ(T, P)$ where any two occurrences in the set are non-overlapping. For any strings X, Y, we say that an occurrence i of a string Z in XY, with $|Z| \geq 2$, *crosses X and Y*, if $i \in [|X| - |Z| + 2 : |X|] \cap Occ(XY, Z)$.

For any strings T and P, we define the sets of *right and left priority non-overlapping occurrences* of P in T, respectively, as follows:

$$RnOcc(T, P) = \begin{cases} \emptyset & \text{if } Occ(T, P) = \emptyset, \\ \{i\} \cup RnOcc(T[1 : i - 1], P) & \text{otherwise,} \end{cases}$$

$$LnOcc(T, P) = \begin{cases} \emptyset & \text{if } Occ(T, P) = \emptyset, \\ \{j\} \cup j + |P| - 1 \oplus LnOcc(T[j + |P| : |T|], P) & \text{otherwise,} \end{cases}$$

where $i = \max Occ(T, P)$ and $j = \min Occ(T, P)$. For all $k \in RnOcc(T, P)$, it is trivially said that $RnOcc(T[k : |T|], P) \subseteq RnOcc(T, P)$. It can be said to $LnOcc$ similarly. Note that $RnOcc(T, P) \subseteq Occ(T, P)$, $LnOcc(T, P) \subseteq Occ(T, P)$, and $LnOcc(T, P) = |T| - |P| + 2 \ominus RnOcc(T^R, P^R)$.

Lemma 1. $nOcc(T, P) = |RnOcc(T, P)| = |LnOcc(T, P)|$

Proof. See Appendix.

Lemma 2. *For any strings T and P, and any integer i with $1 \leq i \leq |T|$, let $u_1 = \max LnOcc(T[1 : i - 1], P) + |P| - 1$ and $u_2 = i - 1 + \min RnOcc(T[i : |T|], P)$. Then $nOcc(T, P) = |LnOcc(T[1 : u_1], P)| + nOcc(T[u_1 + 1 : u_2 - 1], P) + |RnOcc(T[u_2 : |T|], P)|$.*

Proof. By Lemma 1 and the definitions of u_1, u_2, $LnOcc$ and $RnOcc$, we have

$nOcc(T, P)$
$= |LnOcc(T[1 : u_1], P)| + |LnOcc(T[u_1 + 1 : |T|], P)|$
$= |LnOcc(T[1 : u_1], P)| + |RnOcc(T[u_1 + 1 : |T|], P)|$
$= |LnOcc(T[1 : u_1], P)| + |RnOcc(T[u_1 + 1 : u_2 - 1], P)| + |RnOcc(T[u_2 : |T|], P)|$
$= |LnOcc(T[1 : u_1], P)| + nOcc(T[u_1 + 1 : u_2 - 1], P) + |RnOcc(T[u_2 : |T|], P)|$.

$\square$

We will later make use of the solution to the following problem, where occurrences of q-grams are weighted and allowed to overlap.

Problem 1 (weighted overlapping q-gram frequencies). Given a string T, an integer q, and integer array w ($|w| = |T|$), compute $\sum_{i \in Occ(T,P)} w[i]$ for all q-grams $P \in \Sigma^q$ where $Occ(T,P) \neq \emptyset$.

Theorem 1 ([6]). *Problem 1 can be solved in $O(|T|)$ time.*

Proof. See Appendix.

2.3 Straight Line Programs

In this paper, we treat strings described in terms of *straight line programs* (*SLPs*). A straight line program $\mathcal{T}$ is a sequence of assignments $\{X_1 = expr_1, X_2 = expr_2, \ldots, X_n = expr_n\}$. Each X_i is a variable and each $expr_i$ is an expression where $expr_i = a$ ($a \in \Sigma$), or $expr_i = X_\ell X_r$ ($\ell, r < i$). We will sometimes abuse notation and denote $\mathcal{T}$ as $\{X_i\}_{i=1}^n$. Denote by T the string derived from the last variable X_n of the program $\mathcal{T}$. Fig. 1 shows an example of an SLP. The *size* of the program $\mathcal{T}$ is the number n of assignments in $\mathcal{T}$.

Let $val(X_i)$ represent the string derived from X_i. When it is not confusing, we identify a variable X_i with $val(X_i)$. Then, $|X_i|$ denotes the length of the string X_i derives, and $X_i[j] = val(X_i)[j]$, $X_i[j:k] = val(X_i)[j:k]$ for $1 \leq j, k \leq |X_i|$. Let $vOcc(X_i)$ denote the number of times a variable X_i occurs in the derivation of T. For example, $vOcc(X_4) = 3$ in Fig. 1.

Both $|X_i|$ and $vOcc(X_i)$ can be computed for all $1 \leq i \leq n$ in a total of $O(n)$ time by a simple iteration on the variables: $|X_i| = 1$ for any $X_i = a$ ($a \in \Sigma$), and $|X_i| = |X_\ell| + |X_r|$ for

Fig. 1. The derivation tree of SLP $\mathcal{T} = \{X_1 = \mathtt{a}, X_2 = \mathtt{b}, X_3 = X_1 X_2, X_4 = X_1 X_3, X_5 = X_3 X_4, X_6 = X_4 X_5, X_7 = X_6 X_5\}$, which represents string $T = val(X_7) = \mathtt{aababaababaab}$.

any $X_i = X_\ell X_r$. Also, $vOcc(X_n) = 1$ and for $i < n$, $vOcc(X_i) = \sum\{vOcc(X_k) \mid X_k = X_\ell X_i\} + \sum\{vOcc(X_k) \mid X_k = X_i X_r\}$.

We shall assume as in various previous work on SLP, that the word size is at least $\log |T|$, and hence, values representing lengths and positions of T in our algorithms can be manipulated in constant time.
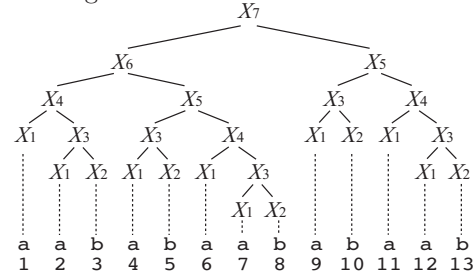
3 q-gram Non-Overlapping Frequencies on Compressed String

The goal of this paper is to efficiently solve the following problem.

Problem 2 (Non-overlapping q-gram frequencies on SLP). Given an SLP $\mathcal{T}$ of size n that describes string T and a positive integer q, compute $nOcc(T,P)$ for all q-grams $P \in \Sigma^q$.

If we decompress the given SLP $\mathcal{T}$ obtaining the string T, then we can solve the problem in $O(|T|)$ time. However, it holds that $|T| = O(2^n)$. Hence, in order to solve the problem efficiently, we have to establish an algorithm that does not explicitly decompress the given SLP $\mathcal{T}$.

3.1 Key Ideas

For any variable X_i and integer $k \geq 1$, let $pre(X_i, k) = X_i[1 : \min\{k, |X_i|\}]$ and $suf(X_i, k) = X_i[|X_i| - \min\{k, |X_i|\} + 1 : |X_i|]$. That is, $pre(X_i, k)$ and $suf(X_i, k)$ are the prefix and the suffix of $val(X_i)$ of length k, respectively. For all variables X_i, $pre(X_i, k)$ can be computed in a total of $O(nk)$ time and space, as follows:

$$pre(X_i, k) = \begin{cases} val(X_i) & \text{if } |X_i| \leq k, \\ pre(X_\ell, k)pre(X_r, k - |X_\ell|) & \text{if } X_i = X_\ell X_r \text{ and } |X_\ell| < k < |X_i|, \\ pre(X_\ell, k) & \text{if } X_i = X_\ell X_r \text{ and } k \leq |X_\ell|. \end{cases}$$

$suf(X_i, k)$ can be computed similarly in $O(nk)$ time and space.

For any string T and positive integers q and j ($1 \leq j \leq j + q - 1 \leq |T|$), the *longest overlapping cover* of the q-gram $P = T[j : j + q - 1]$ w.r.t. position j of T is an ordered pair $\overleftrightarrow{loc}_q(T, j) = (b, e)$ of positions in T which is defined as:

$$\overleftrightarrow{loc}_q(T, j) = \arg\max_{(b,e)} \left\{ (e - b) \; \middle| \; \begin{array}{l} (b, e) \in Occ(T, P) \times ((q - 1) \oplus Occ(T, P)), \\ b \leq j \leq j + q - 1 \leq e, \\ \forall k \in [b : e - q] \cap Occ(T, P), \\ \quad [k + 1 : \min\{k + q - 1, e - q + 1\}] \cap Occ(T, P) \neq \emptyset \end{array} \right\}$$

Namely, $\overleftrightarrow{loc}_q(T, j)$ represents the beginning and ending positions of the maximum chain of overlapping occurrences of q-gram $T[j : j + q - 1]$ that contains position j. For example, consider string $T = \texttt{aaabaabaaabaabaaaabaa}$ of length 21. For $q = 5$ and $j = 9$, we have $\overleftrightarrow{loc}_q(T, j) = (2, 16)$, since $T[2 : 6] = T[5 : 9] = T[9 : 13] = T[12 : 16] = \texttt{aabaa}$. Note that $T[17 : 21] = \texttt{aabaa}$ is not contained in this chain since it does not overlap with $T[12 : 16]$.

Lemma 3. *Given a string T and integers q, j, the longest overlapping cover $\overleftrightarrow{loc}_q(T, j)$ can be computed in $O(|T|)$ time.*

Proof. Using, for example, the KMP algorithm [12], we can obtain a sorted list of $Occ(T, T[j : j + q - 1])$ in $O(|T|)$ time. We can just scan this list forwards and backwards, to easily obtain b and e. $\square$

For a variable $X_i = X_\ell X_r$ and a position $1 \leq j \leq |X_i| - q + 1$, a longest overlapping cover $(b, e) = \overleftrightarrow{loc}_q(X_i, j)$ is said to be *closed in* X_i if $q - 1 < b$ and $e < |X_i| - q + 2$.

Theorem 2. *Problem 2 can be solved in $O(q^2n)$ time, provided that, for all variables $X_i = X_\ell X_r$ and j s.t. $|X_i| \geq q$ and $\max\{1, |X_\ell| - 2(q-1) + 1\} \leq j \leq \min\{|X_\ell| + q - 1, |X_i| - q + 1\}$, $(b, e) = \overleftrightarrow{loc}_q(X_i, j)$ and $nOcc(X_i[b : e], s)$ are already computed where $s = X_i[j : j + q - 1]$.*

Proof. Algorithm 1 shows a pseudo-code of our algorithm to solve Problem 2.

Consider q-gram $s = X_i[j : j + q - 1]$ at position j for which $(b, e) = \overleftrightarrow{loc}_q(X_i, j)$ is closed in X_i. A key observation is that, if (b, e) is closed in X_i, then (b, e) is never closed in X_ℓ or X_r. Therefore, by summing up $vOcc(X_i) \cdot nOcc(X_i[b : e], s)$ for each closed (b, e) in X_i, for all such variables X_i, we obtain $nOcc(T, s)$. Line 14 is sufficient to check if (b, e) is closed.

For all $1 \leq i \leq n$, $vOcc(X_i)$ can be computed in $O(n)$ time, and $t_i = pre(X_i, 2(q-1))suf(X_i, 2(q-1))$ can be computed in $O(qn)$ time and space. The problem amounts to summing up the values of $vOcc(X_i) \cdot nOcc(X_i[b : e], s)$ for each q-gram s contained in each t_i, and can be reduced to Problem 1 on string z and integer array w of length $O(qn)$, which can be solved in $O(qn)$ time by Theorem 1.

In line 15, we check if there is no previous position h ($\max\{1, |X_\ell| - 2(q-1) + 1\} \leq h < j$) such that $X_i[h : h + q - 1] = X_i[j : j + q - 1]$ by $\overrightarrow{loc}_q(X_i, h) = \overrightarrow{loc}_q(X_i, j)$, so that we do not count the same q-gram more than once. If there is no such h, we set the value of $w_i[k - |X_\ell| + j]$ to $vOcc(X_i) \cdot nOcc(X_i[b : e], s)$. This can be checked in $O(q^2n)$ time for all X_i and j.

For convenience, we assume that $T = val(X_n)$ starts and ends with special characters $\#^{q-1}$ and $\$^{q-1}$ that do not occur anywhere else in T, respectively. Then we can cope with the last variable X_n as described above. Hence the theorem holds. □

3.2 Computing Longest Overlapping Covers

In this subsection, we will show how to compute longest overlapping cover $(b, e) = \overleftrightarrow{loc}_q(X_i, j)$ where $s = X_i[j : j + q - 1]$ for all X_i and all j required for Theorem 2.

For any string T and integers q and j ($1 \leq j < q$), let

$$\overrightarrow{loc}_q(T, j) = \begin{cases} (j, be) & \text{if } j + q - 1 \leq |T|, \\ (j, |T|) & \text{otherwise,} \end{cases}$$

$$\overleftarrow{loc}_q(T, j) = \begin{cases} (eb, |T| - j + 1) & \text{if } |T| - j - q + 2 \geq 1, \\ (1, |T| - j + 1) & \text{otherwise,} \end{cases}$$

where $(j, be) = (j - 1) \oplus \overleftarrow{loc}_q(T[j : |T|], 1)$ and $(eb, |T| - j + 1) = \overleftrightarrow{loc}_q(T[1 : |T| - j + 1], |T| - j - q + 2)$. Namely, $\overrightarrow{loc}_q(T, j)$ is a suffix of the longest overlapping cover of the q-gram $T[j : j + q - 1]$ that begins at position j ($1 \leq j < q$) in T, and $\overleftarrow{loc}_q(T, j)$ is a prefix of the longest overlapping cover of the q-gram $T[|T| - j - q + 2 : |T| - j + 1]$ that ends at position $|T| - j + 1$ in T.

Algorithm 1: Computing q-gram non-overlapping frequencies from SLP

Input: SLP $\mathcal{T} = \{X_i\}_{i=1}^n$ representing string T, integer $q \geq 2$.

Output: $nOcc(T, P)$ for all q-grams $P \in \Sigma^q$ where $Occ(T, P) \neq \emptyset$.

1 Compute $vOcc(X_i)$ for all $1 \leq i \leq n$;

2 Compute $pre(X_i, 2(q-1))$ and $suf(X_i, 2(q-1))$ for all $1 \leq i \leq n-1$;

3 $z \leftarrow \varepsilon; w \leftarrow []$;

4 for $i \leftarrow 1$ **to** n **do**

5   **if** $|X_i| \geq q$ **then**

6   let $X_i = X_\ell X_r$;

7 $k \leftarrow |suf(X_\ell, 2(q-1))|$;

8 $t_i = suf(X_\ell, 2(q-1))pre(X_r, 2(q-1))$;

9 z.append(t_i);

10 $w_i \leftarrow$ create integer array of length $|t_i|$, each element set to 0;

11   **for** $j \leftarrow \max\{1, |X_\ell| - 2(q-1) + 1\}$ **to** $\min\{|X_\ell| + q - 1, |X_i| - q + 1\}$ **do**

12 $s \leftarrow X_i[j : j + q - 1]$;

13 $(b, e) \leftarrow \overleftrightarrow{loc}_q(X_i, j)$;

14   **if** $q - 1 < b$ **and** $e < |X_i| - q + 2$ **then**

15   **if** $\overrightarrow{loc}_q(X_i, h) \neq \overrightarrow{loc}_q(X_i, j)$ *for any position* h *s.t.* $\max\{1, |X_\ell| - 2(q-1) + 1\} \leq h < j$ **then**

16 $w_i[k - |X_\ell| + j] \leftarrow vOcc(X_i) \cdot nOcc(X_i[b : e], s)$;

17 w.append(w_i);

18 Calculate q-gram frequencies in z, where each q-gram starting at position d is *weighted* by $w[d]$.

Lemma 4. *For all $1 \leq i \leq n$ and $1 \leq j \leq 2(q-1)$, $\overrightarrow{loc}_q(X_i, j)$ can be computed in a total of $O(q^2 n)$ time.*

Proof. We use dynamic programming. Let $X_i = X_\ell X_r$, $p_j = X_i[j : j+q-1]$, and assume $\overrightarrow{loc}_q(X_\ell, j)$ and $\overrightarrow{loc}_q(X_r, j)$ have been calculated for all $1 \leq j \leq 2(q-1)$. We examine the string $X_i[\max\{j, |X_\ell| - q + 2\} : \min\{|X_i|, |X_\ell| + q - 1\}]$ for occurrences of p_j that cross X_ℓ and X_r, obtain its longest overlapping cover (b_i, e_i), and check if it overlaps with $\overrightarrow{loc}_q(X_\ell, j)$. Furthermore, let bb_r be the left most occurrence of p_j in X_r that has the possibility of overlapping with (b_i, e_i). Then, $\overrightarrow{loc}_q(X_i, j)$ is either $\overrightarrow{loc}_q(X_\ell, j)$, or its end can be extended to e_i, or further to the end of $\overrightarrow{loc}_q(X_r, bb_r)$, depending on how the covers overlap.

More precisely, let $(j, be_\ell) = \overrightarrow{loc}_q(X_\ell, j)$, $(b_i, e_i) = \max\{j - 1, |X_\ell| - q + 1\} \oplus \overleftrightarrow{loc}_q(X_i[\max\{j, |X_\ell| - q + 2\} : \min\{|X_i|, |X_\ell| + q - 1\}], h)$ where $h \in Occ(X_i[\max\{j, |X_\ell| - q + 2\} : \min\{|X_i|, |X_\ell| + q - 1\}], p_j)$, and $(bb_r, be_r) = (|X_\ell| + k - 1) \oplus \overrightarrow{loc}_q(X_r, k)$ where $k = \min Occ(pre(X_r, 2(q-1)), p_j)$. (Note that $(bb_r, be_r), (b_i, e_i)$ are not defined if occurrences h, k of p_j do not exist.) Then we

7

have

$$\overrightarrow{loc}_q(X_i, j) = \begin{cases} (j, be_\ell) & \text{if } be_\ell < b_i \text{ or } \not\exists h, \\ (j, e_i) & \text{if } b_i \leq be_\ell \text{ and } (e_i < bb_r \text{ or } \not\exists k) \\ (j, be_r) & \text{otherwise.} \end{cases}$$

(See also Fig. 2 in Appendix.) For all variables X_i we pre-compute $pre(X_i, 2(q-1))$ and $suf(X_i, 2(q-1))$. This can be done in a total of $O(qn)$ time. Then, each $\overrightarrow{loc}_q(X_i, j)$ can be computed in $O(q)$ time using the KMP algorithm, Lemma 3, and the above recursion, giving a total of $O(q^2 n)$ time for all $1 \leq i \leq n$ and $1 \leq j \leq 2(q-1)$. □

Lemma 5. *For all $1 \leq i \leq n$ and $1 \leq j \leq 2(q-1)$, $\overleftarrow{loc}_q(X_i, j)$ can be computed in a total of $O(q^2 n)$ time.*

Proof. The proof is essentially the same as the proof for $\overrightarrow{loc}_q(X_i, j)$ in Lemma 4.

Recall that we have assumed in Theorem 2 that $\overleftrightarrow{loc}_q(X_i, j)$ are already computed. The following lemma describes how $\overleftrightarrow{loc}_q(X_i, j)$ can actually be computed in a total of $O(q^2 n)$ time.

Lemma 6. *For all variable $X_i = X_\ell X_r$ and j s.t. $\max\{1, |X_\ell| - 2(q-1) + 1\} \leq j \leq \min\{|X_\ell| + q - 1, |X_i| - q + 1\}$, $(b, e) = \overleftrightarrow{loc}_q(X_i, j)$ can be computed in a total of $O(q^2 n)$ time.*

Proof. Let $s_j = X_i[j : j+q-1]$. Firstly, we compute $(b_i, e_i) = \overleftrightarrow{loc}_q(X_i[|X_\ell| - 2(q-1) + 1 : \min\{|X_i|, |X_\ell| + 2(q-1)\}], j)$ and then $\overleftrightarrow{loc}_q(X_i, j)$ can be computed based on (b_i, e_i), as follows: Let $(eb_\ell, ee_\ell) = \overleftarrow{loc}_q(X_\ell, |X_\ell| - ee_\ell + 1)$ and $(bb_r, be_r) = |X_\ell| \oplus \overrightarrow{loc}_q(X_r, bb_r - |X_\ell|)$, where $ee_\ell = \max Occ(X_i[\max\{1, |X_\ell| - 2(q-1) + 1\} : |X_\ell|], s_j)$ and $bb_r = \min Occ(X_i[|X_\ell| + 1 : \min\{|X_i|, |X_\ell| + 2(q-1)\}], s_j)$.

1. If $b_i \leq |X_\ell|$ and $e_i > |X_\ell|$, then we have $b \leq b_i \leq |X_\ell| < e_i \leq e$. $(b, e) = \overleftrightarrow{loc}_q(X_i, j)$ can be computed by checking whether (eb_ℓ, ee_ℓ), (b_i, e_i), and (bb_r, be_r) are overlapping or not. (See also Fig. 3 in Appendix.)
2. If $e_i \leq |X_\ell|$, then trivially $b = eb_\ell$ and $e = e_i$.
3. If $b_i > |X_\ell|$, then trivially $b = b_i$ and $e = be_r$.

Each $ee_\ell = h$ and $bb_r = |X_\ell| + k$ can be computed using the KMP algorithm on string $suf(X_\ell, 2(q-1)) pre(X_r, 2(q-1))$ in $O(q)$ time. By Lemmas 4 and 5, (eb_ℓ, ee_ℓ) and (bb_r, be_r) can be pre-computed in a total of $O(q^2 n)$ time for all $1 \leq i \leq n$. Hence the lemma holds. □

3.3 Largest Left-Priority and Smallest Right-Priority Occurrences

In order to compute $nOcc(X_i[b : e], s)$ for all X_i and all j required for Theorem 2, where $(b, e) = \overleftrightarrow{loc}_q(X_i, j)$ and $s = X_i[j : j + q - 1]$, we will use the largest

8

and second largest occurrences of $LnOcc$ and the smallest and second smallest occurrences of $RnOcc$.

For any set S of integers and integer $1 \leq k \leq |S|$, let $\max_k S$ and $min_k S$ denote the k-th largest and the k-th smallest element of S.

For $1 \leq i \leq n$ and $1 \leq j \leq 2(q-1)$, consider to compute $\max_k LnOcc(X_i[j : be_i], p_j)$ for $k = 1, 2$, where $(j, be_i) = \overrightarrow{loc}_q(X_i, j)$ and $p_j = X_i[j : j + q - 1]$. Intuitively, difficulties in computing $\max_k LnOcc(X_i[j : be_i], p_j)$ come from the fact that the string $val(X_i)[j : be_i]$ can be as long as $O(2^n)$, but we only have prefix $pre(X_i, 3(q-1))$ and suffix $suf(X_i, 3(q-1))$ of $val(X_i)$ of length $O(q)$. Hence we cannot compute the value of be_i by simply running the KMP algorithm on those partial strings. For the same reason, the size of $LnOcc(X_i[j : be_i], p_j)$ can be as large as $O(2^n/q)$. Hence we cannot store $LnOcc(X_i[j : be_i], p_j)$ as is. Still, as will be seen in the following lemma, we can compute those values efficiently, only in $O(q^2 n)$ time.

Lemma 7. *For all variable $X_i = X_\ell X_r$ and $1 \leq j \leq 2(q-1)$, let $(j, be_i) = \overrightarrow{loc}_q(X_i, j)$, $p_j = X_i[j : j + q - 1]$.*
We can compute the values $\max_1 LnOcc(X_i[j : be_i], p_j)$ and $\max_2 LnOcc(X_i[j : be_i], p_j)$
for all $1 \leq i \leq n$ and $1 \leq j \leq 2(q-1)$, in a total of $O(q^2 n)$ time.

Proof. See Appendix.

The next lemma can be shown similarly to Lemma 7.

Lemma 8. *For all variable $X_i = X_\ell X_r$ and $1 \leq j \leq 2(q-1)$, let $(eb, ee) = \overleftarrow{loc}_q(X_i, j)$, and $s_j = X_i[|X_i| - j - q + 2 : |X_i| - j + 1]$. We can compute the values $\min_1 RnOcc(X_i[eb : ee], s_j)$ and $\min_2 RnOcc(X_i[eb : ee], s_j)$ for all $1 \leq i \leq n$ and $1 \leq j \leq 2(q-1)$, in a total of $O(q^2 n)$ time.*

Lemma 9. *For all variable $X_i = X_\ell X_r$ and $1 \leq j < q$, $\max LnOcc(X_i[eb_i : ee_i], s_j)$ can be computed in a total of $O(q^2 n)$ time, where $(eb_i, ee_i) = \overleftarrow{loc}_q(X_i, j)$ and $s_j = X_i[|X_i| - j - q + 2 : |X_i| - j + 1]$.*

Proof. The lemma can be shown by using Lemma 7. See Appendix for details.

Lemma 10. *For all variable $X_i = X_\ell X_r$ and $1 \leq j < q$, $\min RnOcc(X_i[bb_i : be_i], p_j)$ can be computed in a total of $O(q^2 n)$ time, where $(bb_i, be_i) = \overrightarrow{loc}_q(X_i, j)$ and $p_j = X_i[j : j + q - 1]$.*

Proof. The lemma can be shown in a similar way to Lemma 9, using Lemma 8 instead of Lemma 7. $\square$

3.4 Counting Non-Overlapping Occurrences in Longest Overlapping Covers

Firstly, we show how to count non-overlapping occurrences of q-gram p_j in $X_i[j : be_i]$, for all i and j, where $p_j = X_i[j : j + q - 1]$ and $(j, be_i) = \overrightarrow{loc}_q(X_i, j)$.

Lemma 11. *For all variable $X_i = X_\ell X_r$ and $1 \leq j \leq 2(q-1)$, let $(j, be_i) = \overrightarrow{loc}_q(X_i, j)$ and $p_j = X_i[j : j + q - 1]$. We can compute $nOcc(X_i[j : be_i], p_j)$ for all $1 \leq i \leq n$ and $1 \leq j \leq 2(q-1)$, in a total of $O(q^2 n)$ time.*

Proof. By Lemma 1, we have $nOcc(X_i[j : be_i], p_j) = |LnOcc(X_i[j : be_i], p_j)|$. We compute the occurrence b_i in $(j-1) \oplus LnOcc(X_i[j : be_i], p_j)$ that crosses X_ℓ and X_r, if such exists. Note that at most one such occurrence exists. Also, we compute the smallest occurrence bb_r in $(j-1) \oplus LnOcc(X_i[j : be_i], p_j)$ that is completely within X_r. Then the desired value $nOcc(X_i[j : be_i], p_j)$ can be computed depending whether b_i and bb_r exist or not.

Formally: Consider the set $S = ((j-1) \oplus LnOcc(X_i[j : be_i], p_j)) \cap [|X_\ell| - q + 2 : |X_\ell|]$ of occurrence of p_j which is either empty or singleton. If S is singleton, then let b_i be its single element. Let $bb_r = \min\{k \mid k \in ((j-1) \oplus LnOcc(X_i[j : be_i], p_j)) \cap [|X_\ell| + 1 : |X_\ell| + q - 1], \text{if } \exists b_i \text{ then } k \geq b_i + q\}$.

Then we have

$$
nOcc(X_i[j : be_i], p_j)
$$
$$
= \begin{cases}
nOcc(X_r[j - |X_\ell| : be_i - |X_\ell|], p_j) & \text{if } j > |X_\ell|, \\
nOcc(X_\ell[j : be_\ell], p_j) & \text{if } \not\exists b_i \text{ and } \not\exists bb_r, \\
nOcc(X_\ell[j : be_\ell], p_j) + 1 & \text{if } \exists b_i \text{ and } \not\exists bb_r \\
nOcc(X_\ell[j : be_\ell], p_j) + nOcc(X_r[b_r : be_r], p_j) & \text{if } \not\exists b_i \text{ and } \exists bb_r, \\
nOcc(X_\ell[j : be_\ell], p_j) + nOcc(X_r[b_r : be_r], p_j) + 1 & \text{if } \exists b_i \text{ and } \exists bb_r,
\end{cases}
$$

where $(bb_r, be_r) = \overrightarrow{loc}_q(X_r, bb_r)$.

For all variables X_i we pre-compute $pre(X_i, 3(q-1))$ and $suf(X_i, 3(q-1))$. This can be done in a total of $O(qn)$ time. If b_i or bb_r exists, $|X_\ell| - 3(q-1) < j - 1 + \max LnOcc(X_\ell[j : be_\ell], j) \leq |X_\ell| - q + 2$. Then, each b_i and bb_r can be computed from $LnOcc(X_i[(j-1+\max LnOcc(X_\ell[j : be_\ell], j)) : |X_\ell| + 3(q-1)], p_j)$ running the KMP algorithm on string $suf(X_\ell, 3(q-1))pre(X_r, 3(q-1))$. Based on the above recursion, we can compute $nOcc(X_i[j : be_i], p_j)$ in a total of $O(q^2 n)$ time for all $1 \leq i \leq n$ and $1 \leq j \leq 2(q-1)$. □

The next lemma can be shown similarly to Lemma 11.

Lemma 12. *For all variable $X_i = X_\ell X_r$ and $1 \leq j \leq 2(q-1)$, let $(eb_i, ee_i) = \overleftarrow{loc}_q(X_i, j)$ and $s_j = X_i[|X_i| - j - q + 2 : |X_i| - j + 1]$. We can compute $nOcc(X_i[eb_i : ee_i], s_j)$ for all $1 \leq i \leq n$ and $1 \leq j \leq 2(q-1)$, in a total of $O(q^2 n)$ time.*

We have also assumed in Theorem 2 that $nOcc(X_i[b : e], s_j)$ are already computed. This can be computed efficiently, as follows:

Lemma 13. *For all variable $X_i = X_\ell X_r$ and j s.t. $\min\{1, |X_\ell| - 2(q-1) + 1\} \leq j \leq \min\{|X_i| - q + 1, |X_\ell| + q - 1\}$, $nOcc(X_i[b : e], s_j)$ can be computed in a total of $O(q^2 n)$ time, where $(b, e) = \overleftrightarrow{loc}_q(X_i, j)$ and $s_j = X_i[j : j + q - 1]$.*

Proof. We consider the case where $\max\{1, |X_\ell| - q + 2\} \leq j \leq |X_\ell|$, as the other cases can be shown similarly. Our basic strategy for computing $nOcc(X_i[b:e], s_j)$ is as follows. Firstly we compute the largest element of $LnOcc(X_i[b:e], s_j)$ that occurs completely within X_ℓ. Secondly we compute the smallest element of $RnOcc(X_i[b:e], s_j)$ that occurs completely within X_r. Thirdly we compute an occurrence of s_j that crosses the boundary of X_ℓ and X_r, and do not overlap the above occurrences of s_j completely within X_ℓ and X_r.

Formally: Let $ee_\ell = b + q - 2 + \max Occ(X_i[b : |X_\ell|], s_j)$, $bb_r = |X_\ell| + \min Occ(X_i[|X_\ell| + 1 : e], s_j)$, $u_1 = b + q - 2 + \max LnOcc(X_i[b : ee_\ell], s_j)$, and $u_2 = bb_r - 1 + \min RnOcc(X_i[bb_r : e], s_j)$. We consider the case where all these values exist, as other cases can be shown similarly. It follows from Lemmas 1 and 2 that

$$nOcc(X_i[b:e], s_j)$$
$$= |LnOcc(X_i[b:u_1], s_j)| + nOcc(X_i[u_1+1:u_2-1], s_j) + |RnOcc(X_i[u_2:e], s_j)|$$
$$= nOcc(X_i[b:ee_\ell], s_j) + nOcc(X_i[u_1+1:u_2-1], s_j) + nOcc(X_i[bb_r:e], s_j),$$

(See also Fig. 6 in Appendix.)

By Lemma 6, $(b, e) = \overleftrightarrow{loc}_q(X_i, j)$ can be pre-computed in a total of $O(q^2 n)$ time. Since $b < ee_\ell$ and $bb_r < e$, ee_ℓ and bb_r can be computed in $O(q)$ time using the KMP algorithm. By Lemmas 11 and 12 $nOcc(X_i[b : ee_\ell], s_j)$ and $nOcc(X_i[bb_r : e], s_j)$ can be pre-computed in a total of $O(q^2 n)$ time (Notice $(b, ee_\ell) = \overleftarrow{loc}_q(X_\ell, ee_\ell)$ and $(bb_r, e) = |X_\ell| \oplus \overrightarrow{loc}_q(X_r, bb_r - |X_\ell|)$). By Lemmas 9 and 10, u_1 and u_2 can be pre-computed in a total of $O(q^2 n)$ time. Hence $nOcc(X_i[u_1 + 1 : u_2 - 1], s_j)$ can be computed in $O(q)$ time using the KMP algorithm for each i and j. The lemma thus holds. □

3.5 Main Result

The following theorem concludes this whole section.

Theorem 3. *Problem 2 can be solved in $O(q^2 n)$ time and $O(qn)$ space.*

Proof. The time complexity and correctness follow from Theorem 2, Lemma 6, and Lemma 13.

We compute and store strings $suf(X_i, 3(q - 1))$ and $pre(X_i, 3(q - 1))$ of length $O(q)$ for each variable X_i, hence this requires a total of $O(qn)$ space for all $1 \leq i \leq n$. We use a constant number of dynamic programming tables each of which is of size $O(qn)$. Hence the total space complexity is $O(qn)$. □

4 Conclusion and Discussion

We considered the problem of computing the non-overlapping frequencies for all q-grams that occur in a given text represented as an SLP. Our algorithm greatly improves previous work which solved the problem only for $q = 2$ requiring $O(n^4 \log n)$ time and $O(n^3)$ space. We give the first algorithm which works for any $q \geq 2$, running in $O(q^2 n)$ time and $O(qn)$ space, where n is the size of the SLP.

References

1. Amir, A., Benson, G.: Efficient two-dimensional compressed matching. In: Proc. DCC'92. pp. 279–288 (1992)
2. Apostolico, A., Lonardi, S.: Off-line compression by greedy textual substitution. Proceedings of the IEEE 88(11), 1733–1744 (2000)
3. Apostolico, A., Preparata, F.P.: Data structures and algorithms for the string statistics problem. Algorithmica 15(5), 481–494 (1996)
4. Bille, P., Landau, G.M., Raman, R., Sadakane, K., Satti, S.R., Weimann, O.: Random access to grammar-compressed strings. In: Proc. SODA'11. pp. 373–389 (2011)
5. Brodal, G.S., Lyngsø, R.B., Östlin, A., Pedersen, C.N.S.: Solving the string statistics problem in time $O(n \log n)$. In: Proc. ICALP'02. LNCS, vol. 2380, pp. 728–739 (2002)
6. Goto, K., Bannai, H., Inenaga, S., Takeda, M.: Towards efficient mining and classification on compressed strings. In: Accepted for SPIRE'11 (2011), preprint available at arXiv:1103.3114v1
7. Hermelin, D., Landau, G.M., Landau, S., Weimann, O.: A unified algorithm for accelerating edit-distance computation via text-compression. In: Proc. STACS'09. pp. 529–540 (2009)
8. Inenaga, S., Bannai, H.: Finding characteristic substring from compressed texts. In: Proc. The Prague Stringology Conference 2009. pp. 40–54 (2009), full version to appear in the International Journal of Foundations of Computer Science
9. Kärkkäinen, J., Sanders, P.: Simple linear work suffix array construction. In: Proc. ICALP'03. LNCS, vol. 2719, pp. 943–955. Springer (2003)
10. Karpinski, M., Rytter, W., Shinohara, A.: An efficient pattern-matching algorithm for strings with short descriptions. Nordic Journal of Computing 4, 172–186 (1997)
11. Kasai, T., Lee, G., Arimura, H., Arikawa, S., Park, K.: Linear-time Longest-Common-Prefix Computation in Suffix Arrays and Its Applications. In: Proc. CPM'01. LNCS, vol. 2089, pp. 181–192 (2001)
12. Knuth, D.E., Morris, J.H., Pratt, V.R.: Fast pattern matching in strings. SIAM Journal on Computing 6(2), 323–350 (1977)
13. Larsson, N.J., Moffat, A.: Off-line dictionary-based compression. Proceedings of the IEEE 88(11), 1722–1732 (2000)
14. Lifshits, Y.: Processing compressed texts: A tractability border. In: Proc. CPM 2007. LNCS, vol. 4580, pp. 228–240 (2007)
15. Manber, U., Myers, G.: Suffix arrays: A new method for on-line string searches. SIAM Journal on Computing 22(5), 935–948 (1993)
16. Matsubara, W., Inenaga, S., Ishino, A., Shinohara, A., Nakamura, T., Hashimoto, K.: Efficient algorithms to compute compressed longest common substrings and compressed palindromes. Theoretical Computer Science 410(8–10), 900–913 (2009)
17. Navarro, G., Mäkinen, V.: Compressed full-text indexes. ACM Computing Surveys 39(1), 2 (2007)
18. Nevill-Manning, C.G., Witten, I.H., Maulsby, D.L.: Compression by induction of hierarchical grammars. In: Proc. DCC'94. pp. 244–253 (1994)
19. Ziv, J., Lempel, A.: A universal algorithm for sequential data compression. IEEE Transactions on Information Theory IT-23(3), 337–349 (1977)
20. Ziv, J., Lempel, A.: Compression of individual sequences via variable-length coding. IEEE Transactions on Information Theory 24(5), 530–536 (1978)

Appendix

A   Proofs

Proof of Theorem 1.

Proof. We will make use of the *suffix array* and *lcp array*.

The *suffix array* [15] SA of any string T is an array of length $|T|$ such that $SA[i] = j$, where $T[j : |T|]$ is the i-th lexicographically smallest suffix of T. The *lcp* array of any string T is an array of length $|T|$ such that $LCP[i]$ is the length of the longest common prefix of $T[SA[i-1] : |T|]$ and $T[SA[i] : |T|]$ for $2 \leq i \leq |T|$, and $LCP[1] = 0$.

It is well known that the suffix array for any string of length $|T|$ can be constructed in $O(|T|)$ time (e.g. [9]) assuming an integer alphabet. Given the text and suffix array, the lcp array can also be calculated in $O(|T|)$ time [11].

We can calculate the overlapping q-gram frequencies of string T using suffix array SA and lcp array LCP. $SA[i]$ represents an occurrence of a q-gram $T[SA[i] : SA[i] + q - 1]$. Since the suffixes are lexicographically sorted in the suffix array, intervals on the suffix array where the values of lcp array are at least q represent occurrence of the same q-gram. The sum of $w[SA[i]]$ in this interval is the desired value for the q-gram. Constructing SA, LCP can be done in $O(|T|)$ time, and summing up $w[SA[i]]$ for each interval where $LCP[i] \geq q$ can easily be done in $O(|T|)$ by a simple scan. $\qquad\square$

Proof of Lemma 1.

Proof. We prove $nOcc(T[1 : i], P) = |LnOcc(T[1 : i], P)|$ by induction on i. For $i \leq 1$, the statement clearly holds. Now, assume that the statement holds for $i < k$, where $k \geq 2$. For $i = k$, notice that $0 \leq nOcc(T[1 : k], P) - |LnOcc(T[1 : k], P)| \leq 1$, since there can be at most one new occurrence of P ending at position i, which may or may not be counted for $nOcc(T[1 : k], P)$. If we assume on the contrary that the statement does not hold for $i = k$, then $nOcc(T[1 : k], P) - nOcc(T[1 : k - 1], P) = nOcc(T[1 : k], P) - |LnOcc(T[1 : k], P)| = 1$. Since the change was caused by the new occurrence, we have $nOcc(T[1 : k]) = nOcc(T[1 : k - |P|]) + 1$. By the inductive hypothesis, we have $nOcc(T[1 : k - |P|], P) = |LnOcc(T[1 : k - |P|], P)|$. Also, $|LnOcc(T[1 : k], P)| = |LnOcc(T[1 : k - |P|], P)| + 1$, since the new occurrence does not overlap with any occurrences in $LnOcc(T[1 : k - |P|])$. This leads to $nOcc(T[1 : k]) = |LnOcc(T[1 : k], P)|$, a contradiction. $nOcc(T, P) = |RnOcc(T, P)|$ can be shown symmetrically. $\qquad\square$

Proof of Lemma 7.

Proof. We compute the smallest occurrence b_i in $(j-1) \oplus LnOcc(X_i[j : be_i], p_j)$ that crosses X_ℓ and X_r. Also, we compute the smallest occurrence bb_r in $(j-1) \oplus LnOcc(X_i[j : be_i], p_j)$ that is completely within X_r.

13

Then the desired value $\max_1 LnOcc(X_i[j : be_i], p_j)$ can be computed depending whether b_i and bb_r exist or not.

Formally, consider the set $S = ((j-1) \oplus LnOcc(X_i[j : be_i], p_j)) \cap [|X_\ell| - q + 2 : |X_\ell|]$ of occurrence of p_j which is either empty or singleton. If S is singleton, then let b_i be its single element. Let $bb_r = \min\{k \mid k \in ((j-1) \oplus LnOcc(X_i[j : be_i], p_j)) \cap [|X_\ell| + 1 : |X_\ell| + 2(q-1)], \text{if } \exists b_i \text{ then } k \geq b_i + q\}$.

Then we have

$$\max_1 LnOcc(X_i[j : be_i], p_j)$$
$$= \begin{cases} \max_1 LnOcc(X_\ell[j : be_\ell], p_j) & \text{if } \not\exists b_i \text{ and } \not\exists bb_r \\ b_i - j + 1 & \text{if } \exists b_i \text{ and } \not\exists bb_r \\ bb_r - j + \max_1 LnOcc(X_r[bb_r - |X_\ell| : be_r], p_j) & \text{if } \exists bb_r \end{cases}$$

(See also Fig. 7 in Appendix B.)

For all variables X_i we pre-compute $pre(X_i, 3(q-1))$ and $suf(X_i, 3(q-1))$. This can be done in a total of $O(qn)$ time. If b_i or bb_r exists, $|X_\ell| - 3(q-1) \leq j - 1 + \max LnOcc(X_\ell[j : be_\ell], j) \leq |X_\ell| - q + 1$. Then, each b_i and bb_r can be computed from $LnOcc(X_i[(j-1+\max LnOcc(X_\ell[j : be_\ell], j)) : |X_\ell| + 3(q-1)], p_j)$ runnning the KMP algorithm on string $pre(X_i, 3(q-1))suf(X_i, 3(q-1))$.

Based on the above recursion, we can compute $\max_1 LnOcc(X_i[j : be_i], p_j)$ in a total of $O(q^2 n)$ time for all $1 \leq i \leq n$ and $1 \leq j \leq 2(q-1)$.

It is not difficult to see that similar claims, with slightly different conditions, can be made for $\max_2 LnOcc(X_i[j : be_i], p_j)$ where the value corresponds to one of 4 values: $\max_2 LnOcc(X_\ell[j : be_\ell], p_j)$, $\max_1 LnOcc(X_\ell[j : be_\ell], p_j)$, b_i, or $\max_2 LnOcc(X_r[bb_r - |X_\ell| : be_r], p_j)$, with appropriate offsets. $\square$

Proof of Lemma 9.

Proof. Our basic strategy for computing $\max LnOcc(X_i[eb_i : ee_i], s_j)$ is as follows. Firstly we compute the largest element of $LnOcc(X_i[eb_i : ee_i], s_j)$ that occurs completely within X_ℓ. Secondly we compute the smallest element of $LnOcc(X_i[eb_i : ee_i], s_j)$ that crosses the boundary of X_ℓ and X_r. Let d be this occurrence, if such exists. Then the desired output $\max LnOcc(X_i[eb_i : ee_i], s_j)$ is given as either the largest or the second largest element of $(d + q - 1) \oplus LnOcc(X_r[d + q - |X_\ell| : |X_r|], s_j)$.

More formally: We consider the case where $eb_i + q - 1 \leq |X_\ell|$. Let $ee_\ell = q - 1 + \max(Occ(X_i, s_j) \cap [|X_\ell| - 2(q-1) + 1 : \overleftarrow{|X_\ell| - q + 1}])$, $m = eb_i - 1 + \max LnOcc(X_\ell[eb_i : ee_\ell], s_j)$ where $(eb_i, ee_\ell) = \overleftarrow{loc}_q(X_\ell, |X_\ell| - ee_\ell + 1)$. Let $d = m + q - 1 + \min LnOcc(X_i[m + q : ee_i], s_j)$. Let

$$bb_r = \begin{cases} d & \text{if } ee_i - q + 1 \leq |X_\ell| \text{ or } d > |X_\ell|, \\ d + q - 1 + \min LnOcc(X_i[d + q : |X_i|], s_j) & \text{otherwise.} \end{cases}$$

Let $h' = |X_\ell| + \max_2 LnOcc(X_r[bb_{r'} : be_{r'}], s_j)$ and $h = |X_\ell| + \max_1 LnOcc(X_r[bb_{r'} : be_{r'}], s_j)$ where $(bb_{r'}, be_{r'}) = \overrightarrow{loc}_q(X_r, bb_r - |X_\ell|)$. (See also Fig. 5 in Appendix B.)

14

Then

$$\max LnOcc(X_i[eb_i : ee_i], s_j) = \begin{cases} h & \text{if } h \leq ee_i - q + 1, \\ h' & \text{otherwise.} \end{cases}$$

The case where $eb_i + q - 1 > |X_\ell|$ can be solved similarly.

Each ee_ℓ, d and bb_r can be computed in $O(q)$ time using the KMP algorithm, hence requiring a total of $O(q^2 n)$ time. By Lemmas 4 and 5, $\overleftarrow{loc}_q(X_\ell, ee_\ell)$ and $\overrightarrow{loc}_q(X_i, bb_r)$ can be computed in $O(q^2 n)$ time for all $X_i = X_\ell X_r$ and $1 \leq j < n$. By Lemma 7, h' and h can be computed in a total of $O(q^2 n)$ time for all $X_i = X_\ell X_r$ and $1 \leq j < n$. Therefore, by dynamic programming we can compute $LnOcc(X_i[eb_i : ee_i], s_j)$ in a total of $O(q^2 n)$ time. $\qquad \square$

B Figures

Fig. 2. Illustration for Lemma 4. In this figure, $\overrightarrow{loc}_q(X_i, j) = (j, e_i)$.
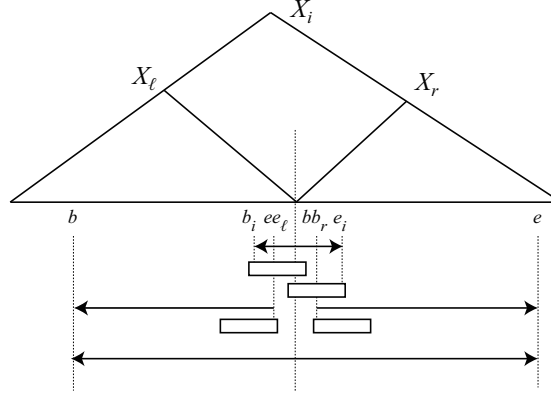

Fig. 3. Illustration for Lemma 6. Rectangles show important occurrences of $X_i[j : j + q - 1]$. In this case $b = eb_\ell$ and $e = be_r$.
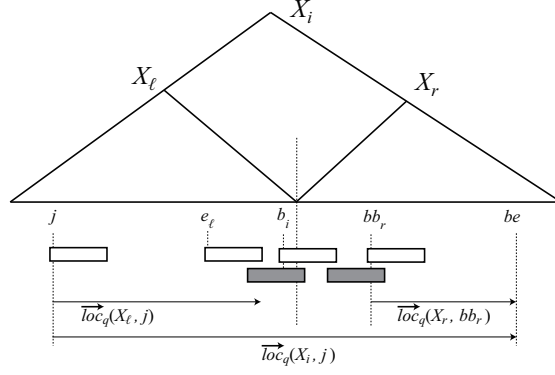
Fig. 4. Illustration for Lemma 7, calculating max $LnOcc(X_i[j : be], p_j)$. Shadowed occurrences are not in $LnOcc(X_i[j : be_i], p_j)$, while white ones are in $LnOcc(X_i[j : be_i], p_j)$.
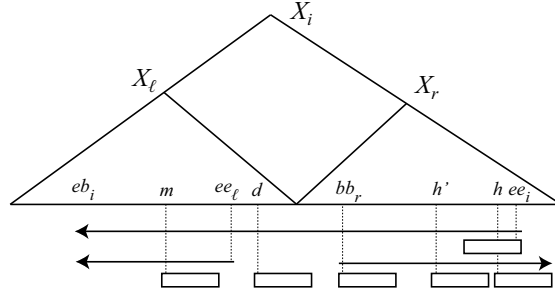

Fig. 5. Illustration for Lemma 9. Rectangles show important occurrences of s_j. In this case max $LnOcc(X_i[eb_i, ee_i], s_j) = h'$, as $h > ee_i - q + 1$.
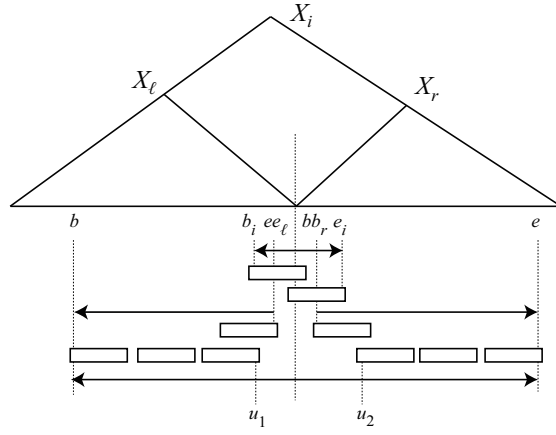
Fig. 6. Illustration for Lemma 13. Rectangles show important occurrences of $X_i[j : j + q - 1]$. In this case $nOcc(X_i[b : ee_\ell], s_j) = 3$, $nOcc(X_i[u_1 + 1 : u_2 - 1], s_j) = 1$, and $nOcc(X_i[bb_r : e], s_j) = 3$.