Finding Characteristic Substrings from Compressed Texts

Shunsuke Inenaga[1] and Hideo Bannai[2]

[1] Graduate School of Information Science and Electrical Engineering, Kyushu University, Japan
inenaga@c.csce.kyushu-u.ac.jp
[2] Department of Informatics, Kyushu University, Japan
bannai@inf.kyushu-u.ac.jp

Abstract. Text mining from large scaled data is of great importance in computer science. In this paper, we consider fundamental problems on text mining from compressed strings, i.e., computing a longest repeating substring, longest non-overlapping repeating substring, most frequent substring, and most frequent non-overlapping substring from a given compressed string. Also, we tackle the following novel problem: given a compressed text and compressed pattern, compute the representative of the equivalence class of the pattern w.r.t. the text. We present algorithms that solve the above problems in time polynomial in the size of input compressed strings. The compression scheme we consider is straight line program (SLP) which has exponential compression, and therefore our algorithms are more efficient than any algorithms that work on uncompressed strings.

1 Introduction

Text mining from large scaled data, e.g. biological and web data, is currently a very important topic in computer science [2]. The sheer size of the data makes the task difficult, and hence, it is convenient to store these data in a compressed form. The question is if it is possible to perform text mining operations on compressed strings.

In this paper, we consider the following fundamental text mining problems from compressed strings: given a compressed form $\mathcal{T}$ of a string T, compute (1) *a longest repeating substring* of T, (2) *a longest non-overlapping repeating substring* of T, (3) *a most frequent substring* of T, (4) *a most frequent non-overlapping substring* of T. We present algorithms to solve Problem 1 in $O(n^4 \log n)$ time and $O(n^3)$ space, Problem 2 in $O(n^6 \log n)$ time and $O(n^3)$ space, Problem 3 in $O(|\Sigma|^2 n^2)$ time and $O(n^2)$ space, and Problem 4 in $O(n^4 \log n)$ time and $O(n^3)$ space, where n is the size of $\mathcal{T}$ and Σ is the alphabet. We also consider the following problem: given compressed forms of two strings T and P, compute *the representative* of the string equivalence class [1] of P in T. We present an $O(n^4 \log n)$-time $O(n^3)$-space algorithm to solve this problem, where n denotes the total size of the two compressed representations. By computing the representative of the equivalence class, we can retrieve the left and right contexts of P in T. The equivalence class and its representative have played central roles in the discovery of characteristic expressions in classical Japanese poems [13], and in a blog spam detection algorithm [10]. To the best of our knowledge, our algorithms are the first to solve the above problems *without decompression*.

The text compression scheme we consider in this paper is *straight line program* (*SLP*). SLP is a context-free grammar in the Chomsky normal form and generates a single string. SLP is an abstract model of many kinds of text compression schemes, as

Shunsuke Inenaga, Hideo Bannai: Finding Characteristic Substrings from Compressed Texts , pp. 40–54.
Proceedings of PSC 2009, Jan Holub and Jan Žďárek (Eds.), ISBN 978-80-01-04403-2 © Czech Technical University in Prague, Czech Republic

the resulting encoding of the LZ-family [14,15], run-length, multi-level pattern matching code [5], Sequitur [11] and so on, can quickly be transformed into SLPs [3,12][1]. The important property of SLP is that it allows *exponential* compression, i.e., the original (uncompressed) string length N can be exponentially large w.r.t. the corresponding SLP size n. Therefore, our algorithms are asymptotically faster than *any* approaches that treat uncompressed strings.

Related Work. Little is known for text mining from compressed strings. Gąsieniec et al. [3] stated that it is possible to compute a succinct representation of all squares that appear in a given compressed string of size n in $O(n^6 \log^5 N)$ time. Matsubara et al. [8] presented an $O(n^4)$-time $O(n^2)$-space algorithm to compute a succinct representation of all maximal palindromes from a given SLP-compressed string.

2 Preliminaries

2.1 Notations

For any set U of integers and an integer k, we denote $U \oplus k = \{i + k \mid i \in U\}$ and $U \ominus k = \{i - k \mid i \in U\}$.

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, $|\varepsilon| = 0$. 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 $j < i$.

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

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

For any two strings T and S, let $LCPref(T, S)$ and $LCSuf(T, S)$ denote the length of the *longest common prefix and suffix* of T and S, respectively.

For any string T, let $\overline{T}$ denote the reversed string of T, i.e., $\overline{T} = T[|T|] \cdots T[2]T[1]$.

A *period* of a string T is an integer p $(1 \leq p \leq |T|)$ such that $T[i] = T[i + p]$ for any $i = 1, 2, \ldots, |T| - p$.

For any two strings T and S, we define the set $OL(T, S)$ as follows:

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

Namely, $k \in OL(T, S)$ iff the suffix of T of length k $(k > 0)$ matches the prefix of S of length k.

2.2 Text Compression by 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 such that

$$X_1 = expr_1, X_2 = expr_2, \ldots, X_n = expr_n,$$

where each X_i is a variable and each $expr_i$ is an expression either

- $expr_i = a \quad (a \in \Sigma)$, or
- $expr_i = X_\ell X_r \quad (\ell, r < i)$.

[1] An important exception is compression schemes based on the Burrows-Wheeler transform.

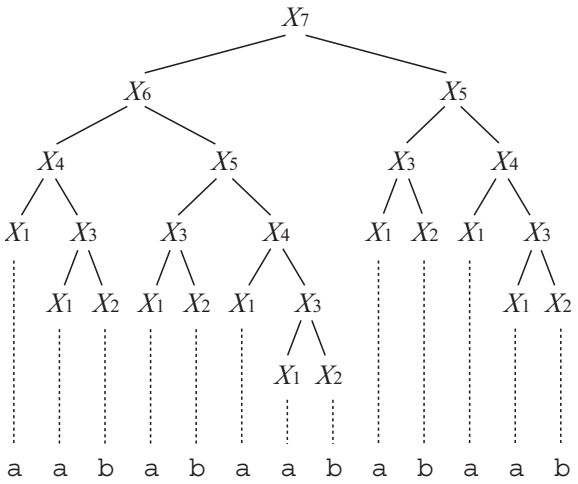

Figure 1. The derivation tree of an SLP that generates string aababaababaab.

Denote by T the string derived from the last variable X_n of the program $\mathcal{T}$. The *size* of the program $\mathcal{T}$ is the number n of assignments in $\mathcal{T}$. We remark that $|T| = O(2^n)$. Figure 1 shows the derivation tree of an SLP which derives string aababaababaab.

When it is not confusing, we identify a variable X_i with the string derived from X_i. Then, $|X_i|$ denotes the length of the string derived from X_i.

For any variable X_i of $\mathcal{T}$ with $1 \leq i \leq n$, we define $\overline{X_i}$ as follows:

$$\overline{X_i} = \begin{cases} a & \text{if } X_i = a \quad (a \in \Sigma), \\ \overline{X_r}\ \overline{X_\ell} & \text{if } X_i = X_\ell X_r \quad (\ell, r < i). \end{cases}$$

Let $\overline{\mathcal{T}}$ be the SLP consisting of variables $\overline{X_i}$ for $1 \leq i \leq n$. It is shown in [8] that SLP $\overline{\mathcal{T}}$ derives string $\overline{T}$ and $\overline{\mathcal{T}}$ can be easily computed from SLP $\mathcal{T}$ in $O(n)$ time.

Pattern Matching on SLP-compressed Strings. Here we briefly recall some existing results on pattern matching for SLP-compressed strings.

Let Y_j denote a variable of SLP $\mathcal{P}$ of size m that generates string P, for $1 \leq j \leq m$.

For any SLP variables $X_i = X_\ell X_r$ and Y_j, we define the set $Occ^{\triangle}(X_i, Y_j)$ of all occurrences of Y_j that cover or touch the boundary between X_ℓ and X_r, namely,

$$Occ^{\triangle}(X_i, Y_j) = \{s > 0 \mid X_i[s : s + |Y_j| - 1] = Y_j, |X_\ell| - |Y_j| + 1 \leq s \leq |X_\ell|\}.$$

Lemma 1 ([9]). *For any SLP variables X_i and Y_j, $Occ^{\triangle}(X_i, Y_j)$ forms a single arithmetic progression. Moreover, if $|Occ^{\triangle}(X_i, Y_j)| \geq 3$, then the common difference of the arithmetic progression is the smallest period of Y_j.*

Lemma 2 ([9]). *A membership query to $Occ(T, P)$ can be answered in $O(n)$ time, provided that $Occ^{\triangle}(X_i, Y_m)$ is already computed for every X_i.*

Theorem 3 ([7]). *$Occ^{\triangle}(X_i, Y_j)$ can be computed in a total of $O(n^2 m)$ time and $O(nm)$ space for every $1 \leq i \leq n$ and $1 \leq j \leq m$, using a DP table App of size $n \times m$ such that $App[i, j]$ stores an arithmetic progression for $Occ^{\triangle}(X_i, Y_j)$.*

Computing Overlaps of SLP-compressed Strings. The set of overlaps between two variables can be efficiently computed as follows (assume $n > m$).

Lemma 4 ([4]). *For any SLP variables X_i and Y_j, $OL(X_i, Y_j)$ can be represented by $O(n)$ arithmetic progressions.*

Theorem 5 ([4]). *$OL(X_i, Y_j)$ can be computed in total of $O(n^4 \log n)$ time and $O(n^3)$ space for every $1 \leq i \leq n$ and $1 \leq j \leq m$.*

2.3 The *FM* function

For any two SLP variables X_i, Y_j and any integer k with $1 \leq k \leq |X_i|$, we define function $FM(X_i, Y_j, k)$ which returns the length of the longest common prefix of $X_i[k : |X_i|]$ and Y_j, that is,

$$FM(X_i, Y_j, k) = LCPref(X_i[k : |X_i|], Y_j).$$

Lemma 6 ([4]). *For any SLP variables X_i, Y_j and integer k, $FM(X_i, Y_j, k)$ can be computed in $O(n \log n)$ time, provided that $OL(X_{i'}, Y_{j'})$ is already computed for any $1 \leq i' \leq i$ and $1 \leq j' \leq j$.*

3 Computing Repeating Substrings from Compressed Text

3.1 Problems

A string P is said to be a *repeating substring* of a string T if $|Occ(T, P)| \geq 2$. A *longest repeating substring* of T is a longest string P of T such that $|Occ(T, P)| \geq 2$. A *most frequent substring* of T is a string P such that $|Occ(T, P)| \geq |Occ(T, Q)|$ for any other string Q.

Any two occurrences $k_1, k_2 \in Occ(T, P)$ with $k_1 < k_2$ are said to *overlap* if $k_1 + |P| \geq k_2$. Otherwise, they are said to *non-overlap*. A *longest non-overlapping repeating substring* of T is a longest string P such that there exist at least two non-overlapping occurrences in $Occ(T, P)$. A *most frequent non-overlapping substring* of T is a string P such that it has the most non-overlapping occurrences in T.

In this section we consider the following problems.

Problem 1 (Computing longest repeating substring from SLP). Given an SLP $\mathcal{T}$ that derives a string T, compute two occurrences of a longest repeating substring P of T and its length $|P|$.

Problem 2 (Computing longest non-overlapping repeating substring from SLP). Given an SLP $\mathcal{T}$ that derives a string T, compute two non-overlapping occurrences of a longest non-overlapping repeating substring P of T and its length $|P|$.

Problem 3 (Computing most frequent substring from SLP). Given an SLP $\mathcal{T}$ that derives a string T, compute a most frequent substring P of T and a representation and the cardinality of $Occ(T, P)$.

Problem 4 (Computing most frequent non-overlapping substring from SLP). Given an SLP $\mathcal{T}$ that derives a string T, compute a most frequent non-overlapping substring P of T, and a representation and the number of non-overlapping occurrences of P in T.

By "representation" in Problems 3 and 4 we mean some succinct (polynomial-sized) representation of the sets. This is due to the fact that the cardinality of the sets can be exponentially large w.r.t. the input size.

In what follows, let n be the size of SLP $\mathcal{T}$ and let X_i denote each variable of $\mathcal{T}$ for $1 \leq i \leq n$.

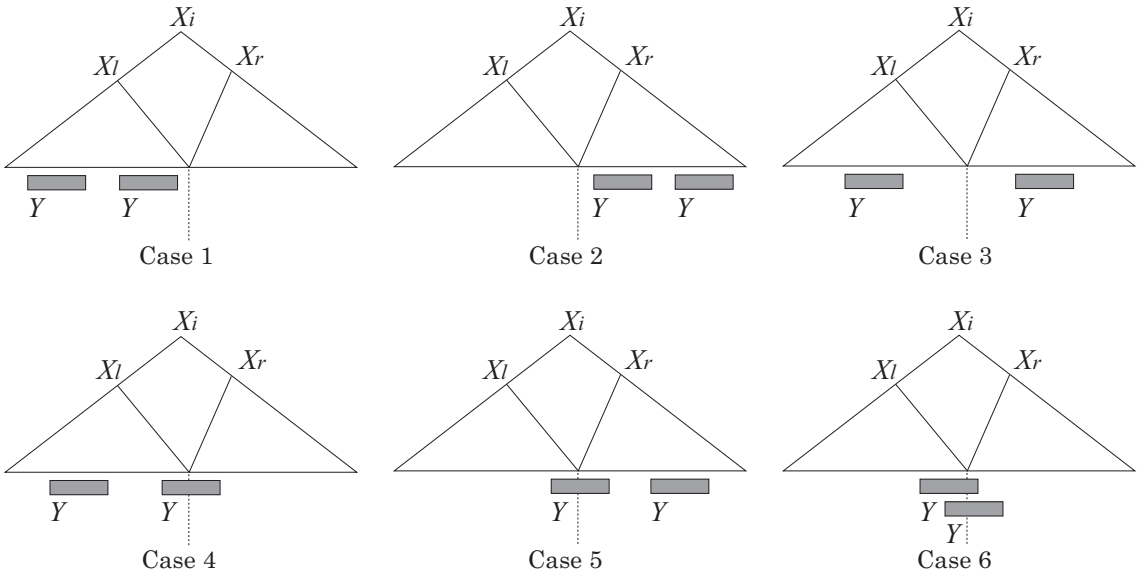

Figure 2. Illustration for Observation 7. The six cases for two distinct occurrences of a substring Y of X_i.

3.2 Solution to Problem 1

A key observation for solving Problem 1 is the following.

Observation 7. *For any SLP variable $X_i = X_\ell X_r$ and any string Y, assume that $|Occ(X_i, Y)| \geq 2$. Any two occurrences $k_1, k_2 \in Occ(X_i, Y)$ with $k_1 < k_2$ fall into one of the six following cases (see also Figure 2):*

1. $k_1, k_2 \in Occ(X_\ell, Y)$.
2. $k_1, k_2 \in Occ(X_r, Y)$.
3. $k_1 \in Occ(X_\ell, Y)$ *and* $k_2 \in Occ(X_r, Y)$.
4. $k_1 \in Occ(X_\ell, Y)$ *and* $k_2 \in Occ^{\triangle}(X_i, Y)$.
5. $k_1 \in Occ^{\triangle}(X_i, Y)$ *and* $k_2 \in Occ(X_r, Y)$.
6. $k_1, k_2 \in Occ^{\triangle}(X_i, Y)$.

Observation 7 implies that a longest repeating substring of T can be obtained by computing a longest repeating substring for every SLP variable X_i in each case. Case 1 and Case 2 are symmetric, and these two cases actually belong to one of the above cases with respect to variables X_ℓ and X_r, respectively. Since Case 4 and Case 5 are symmetric, we focus on Case 3, Case 4, and Case 6 in the sequel.

The following lemma is useful to deal with Case 3.

Lemma 8 ([8]). *For every pair of SLP variables X_i and X_j, we can compute the length of a longest common substring of X_i and X_j plus its occurrence position in X_i and X_j in $O(n^2 \log n)$ time, provided that $OL(X_{i'}, X_{j'})$ is already computed for any $1 \leq i' \leq i$ and $1 \leq j' \leq j$.*

Now we have the next lemma.

Lemma 9. *For every SLP variable X_i, two occurrences and the length of a longest repeating substring in Case 3 can be computed in $O(n^2 \log n)$ time.*

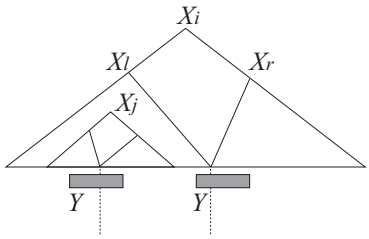

Figure 3. Illustration for Observation 10.

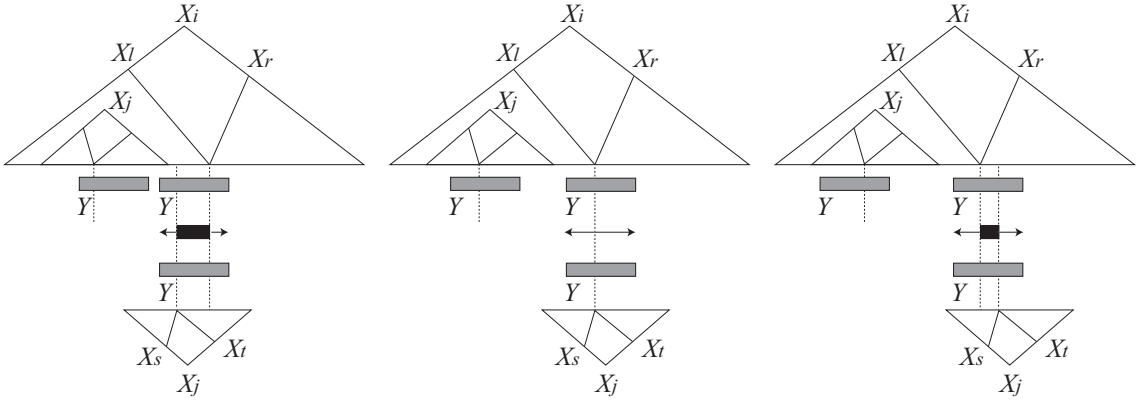

Figure 4. Illustration for Observation 11. The black rectangles of the left and right diagrams are an element of $OL(X_\ell, X_t)$ and $OL(X_s, X_r)$, respectively.

Proof. Note that a longest repeating substring of $X_i = X_\ell X_r$ in Case 3 is indeed a longest common substring of X_ℓ and X_r. Hence the lemma holds by Lemma 8. □

Next we consider Case 4. A key observation is the following:

Observation 10. *For the first occurrence of Y in Case 4 of Observation 7, there always exists a variable X_j such that X_j is a descendant of X_ℓ or X_ℓ itself, and the first occurrence of Y touches or covers the boundary of X_j (see also Figure 3).*

For any SLP variables $X_i = X_\ell X_r$ and $X_j = X_s X_t$ and any non-negative integer $z \in OL(X_\ell, X_t) \cup \{0\}$, let h_1 and h_2 be the maximum non-negative integers such that

$$X_i[|X_\ell| - z - h_1 + 1 : |X_\ell| + h_2] = X_j[|X_s| - h_1 + 1 : |X_s| + z + h_2].$$

That is, $h_1 = LCSuf(X_\ell[1 : |X_\ell| - z], X_s)$ and $h_2 = LCPref(X_r, X_t[z + 1 : |X_t|])$. Let

$$Ext_{X_i,X_j}(z) = \begin{cases} z + h_1 + h_2 & \text{if } X_i = X_\ell X_r \text{ and } X_j = X_s X_t, \\ z & \text{if } X_i \text{ or } X_j \text{ is constant.} \end{cases}$$

For a set S of integers, we define $Ext_{X_i,X_j}(S) = \{Ext_{X_i,X_j}(z) \mid z \in S\}$. $Ext_{X_j,X_i}(z)$ and $Ext_{X_j,X_i}(S)$ are defined similarly.

Observation 11. *The length of a longest repeating substring of Case 4 is equal to the maximum element of*

$$\bigcup_{X_j \in A} (Ext_{X_i,X_j}(OL(X_\ell, X_t)) \cup Ext_{X_i,X_j}(0) \cup Ext_{X_j,X_i}(OL(X_s, X_r))) \tag{1}$$

where $A = \{X_j = X_s X_t \mid X_j$ is a descendant of X_ℓ or $j = \ell\}$. (See also Figure 4.)

Now we have the following lemma.

Lemma 12. *For every SLP variable X_i, two occurrences and the length of a longest repeating substring in Case 4 can be computed in $O(n^3 \log n)$ time, provided that $OL(X_{i'}, X_{j'})$ and $Occ^{\triangle}(X_{i'}, X_{j'})$ are already computed for any $1 \leq i' \leq n$ and $1 \leq j' \leq n$.*

Proof. Let $X_i = X_\ell X_r$. It was proven by Lemma 4 of [8] that, for any variable $X_j = X_s X_t$ mentioned in Observation 10, $\max(Ext_{X_i,X_j}(OL(X_\ell, X_t)) \cup Ext_{X_i,X_j}(0) \cup Ext_{X_j,X_i}(OL(X_s, X_r)))$ can be computed in $O(n^2 \log n)$ time, provided that $OL(X_{i'}, X_{j'})$ is already computed for any $1 \leq i' \leq n$ and $1 \leq j' \leq n$. Recall Observation 11. Since the number of distinct descendants of any variable X_i is at most $n-1$, we can compute Equation (1) in $O(n^3 \log n)$ time. Let X_h be the variable that gives the maximum value of Equation (1). We can retrieve one position of $Occ(X_\ell, X_h)$ in $O(n^2)$ time from $Occ^{\triangle}(X_1, X_h), \ldots, Occ^{\triangle}(X_\ell, X_h)$. Then it is easy to compute two occurrences of the longest repeating substring in constant time. □

It is not difficult to see that Case 6 can be solved in a similar way to Case 4.

By Lemma 9, Lemma 12, Theorem 3, and Theorem 5, we obtain the main result of this subsection.

Theorem 13. *Problem 1 can be solved in $O(n^4 \log n)$ time and $O(n^3)$ space.*

3.3 Solution to Problem 2

Here we show how to find a longest non-overlapping repeating substring from a given SLP. The algorithm is based on the one proposed in Section 3.2. Below, we give our strategy to find a maximal non-overlapping repeating substring from the overlapping repeating substring found by the algorithm of Section 3.2.

An obvious fact is that Case 3 of Observation 7 only deals with a non-overlapping repeating substring. Hence we focus on Case 4. The other cases are solved similarly.

Lemma 14 ([6]). *Let $\mathcal{T}$ be any SLP of size n that generates string T. For any substring Y of T, it takes $O(n)$ time construct a new SLP of size $O(n)$ which generates the substring Y.*

Lemma 15. *Let k_1 and k_2 be any overlapping occurrences of string Y in string X such that $k_1 < k_2$. Let p be the smallest period of Y. Then, a longest non-overlapping repeating substring in $X[k_1 : k_2 + |Y| - 1]$ is $Y[1 : k_2 - 1 + pl]$, where $l = \lfloor (|Y| - k_2 + k_1)/2p \rfloor$.*

Proof. The length of the overlap is $|Y| - k_2 + k_1$. $Y[1 : p]$ appears in $\lfloor (|Y| - k_2 + k_1)/p \rfloor$ times in the overlap part $Y[k_2 : |Y| + k_1]$. Hence the lemma holds. □

Lemma 16. *For every SLP variable X_i, two occurrences and the length of a longest non-overlapping repeating substring in Case 4 of Observation 7 can be computed in $O(n^5 \log n)$ time and $O(n^3)$ space, provided that $OL(X_{i'}, X_{j'})$ is already computed for any $1 \leq i' \leq n$ and $1 \leq j' \leq n$.*

Proof. The proof is based on the proof of Lemma 8 of [8].

We consider $Ext_{X_i,X_j}(OL(X_\ell, X_t))$ of Observation 11. For each descendant X_j of X_i, it is sufficient to consider the leftmost occurrence γ of X_j in the derivation tree

of X_i, since no other occurrences of X_j can correspond to longer non-overlapping repeating substring than the leftmost occurrence.

Let $\langle a, d, q\rangle$ denote any of the $O(n)$ arithmetic progressions in $OL(X_\ell, X_t)$, where a denotes the minimal element, d does the common difference and q does the number of elements of the progression. That is, $\langle a, d, q\rangle = \{a + (i-1)d \mid 1 \le i \le q\}$.

Assume $q > 1$ and $a < d$, as the case where $q = 1$ or $a = d$ is easier to show. Let $u = X_t[1 : a]$ and $v = X_t[a+1 : d]$.

Let e_1, e_2 be the largest integer such that $X_i[|X_\ell| - e_2 + 1 : |X_\ell| + e_1]$ is the longest substring of X_i that contains $X_i[|X_\ell| - d + 1 : |X_\ell|]$ and has a period d. Similarly, let e_3, e_4 be the largest integer such that $X_j[|X_s| - e_4 + 1 : |X_s| + e_3]$ is the longest substring of X_j that contains $X_j[|X_s|+1 : |X_s|+d]$ and has a period d. More formally,

$$e_1 = LCPref(X_r, (vu)^*) = \begin{cases} FM(X_t, X_r, a+1) & \text{if } FM(X_t, X_r, a+1) < d, \\ FM(X_r, X_r, d+1) + d & \text{otherwise,} \end{cases}$$

$$e_2 = LCSuf(X_\ell, (vu)^*) = FM(\overline{X_\ell}, \overline{X_\ell}, d+1) + d,$$

$$e_3 = LCPref(X_t, (uv)^*) = FM(X_t, X_t, d+1) + d,$$

$$e_4 = LCSuf(X_s, (uv)^*) = \begin{cases} FM(\overline{X_\ell}, \overline{X_s}, a+1) & \text{if } FM(\overline{X_\ell}, \overline{X_s}, a+1) < d, \\ FM(\overline{X_s}, \overline{X_s}, d+1) + d & \text{otherwise,} \end{cases}$$

where $(vu)^*$ and $(uv)^*$ denote infinite repetitions of vu and uv, respectively.

Let $k \in \langle a, d, q\rangle$. We categorize $Ext_{X_i,X_j}(k)$ depending on the value of k, as follows.

(1) When $k < \min\{e_3 - e_1, e_2 - e_4\}$. If $k - d \in \langle a, d, q\rangle$, then we have $Ext_{X_i,X_j}(k) = Ext_{X_i,X_j}(k-d) + d$.

(2) When $k > \max\{e_3 - e_1, e_2 - e_4\}$. If $k + d \in \langle a, d, q\rangle$, then we have $Ext_{X_i,X_j}(k) = Ext_{X_i,X_j}(k+d) + d$.

(3) When $\min\{e_3 - e_1, e_2 - e_4\} < k < \max\{e_3 - e_1, e_2 - e_4\}$. In this case we have $Ext_{X_i,X_j}(k) = \min\{e_1 + e_2, e_3 + e_4\}$ for any k with $\min\{e_3 - e_1, e_2 - e_4\} < k < \max\{e_3 - e_1, e_2 - e_4\}$.

(4) When $k = e_3 - e_1$. In this case we have

$$\begin{aligned} Ext_{X_i,X_j}(k) &= k + \min\{e_2 - k, e_4\} + LCPref(X_t[k+1 : |X_t|], X_r) \\ &= k + \min\{e_2 - k, e_4\} + FM(X_t, X_r, k+1). \end{aligned}$$

(5) When $k = e_2 - e_4$. In this case we have

$$\begin{aligned} Ext_{X_i,X_j}(k) &= k + LCSuf(X_\ell[1 : |X_\ell| - k], X_s) + \min\{e_1, e_3 - k\} \\ &= k + FM(\overline{X_\ell}, \overline{X_s}, k+1) + \min\{e_1, e_3 - k\}. \end{aligned}$$

(6) When $k = e_3 - e_1 = e_2 - e_4$. In this case we have

$$\begin{aligned} Ext_{X_i,X_j}(k) &= k + LCSuf(X_\ell[1 : |X_\ell| - k], X_s) + LCPref(X_t[k+1 : |X_t|], X_r) \\ &= k + FM(\overline{X_\ell}, \overline{X_s}, k+1) + FM(X_t, X_r, k+1). \end{aligned}$$

Consider Case (1). For any $k - d, k \in \langle a, d, q\rangle$, if the occurrences of the substring that corresponds to $Ext_{X_i,X_j}(k-d)$ overlap, then the substring that corresponds to $Ext_{X_i,X_j}(k)$ also overlap. Note also that these substrings have the same ending position b in X_i, and have the same beginning position c in X_j. Since a membership query to the triple $\langle a, d, q\rangle$ can be answered in constant time, we can find the largest

k belonging to Case (1) in constant time. It is not difficult to see that d is the smallest period of $X_i[c+\gamma-1:b]$. By Lemma 15, we can compute the length of a longest non-overlapping repeating substring in $X_i[c+\gamma-1:b]$ in constant time, provided that e_1, e_2, e_3, e_4 are already computed. Similar arguments hold for Cases (2) and (3).

Consider Case (4). Let Z be the unique substring that corresponds to $Ext_{X_i,X_j}(k)$. Let x and y be the integers such that $x < y$ and $X_h[x : x+|Z|-1] = X_i[y : y+|Z|-1] = Z$. If $x+\gamma+|Z|-2 \geq y$, then we construct a new SLP $\mathcal{P}$ that generates string $P = X_i[x+\gamma-1 : y+|Z|-1]$ and compute its smallest period. It is clear that $|P| - \max(OL(P,P) - \{|P|\})$ is the smallest period of P. By Theorem 5 and Lemma 14, the length of a longest non-overlapping repeating substring in P can be computed in $O(n^4 \log n)$ time with $O(n^3)$ space. Similar arguments hold for Cases (5) and (6).

The values of e_1, e_2, e_3, e_4 can be computed by at most 6 calls of the FM function, each taking $O(n \log n)$ time.

Since there is $O(n)$ descendants of X_i, the total cost is $O(n^5 \log n)$ time and $O(n^3)$ space. □

The next theorem follows from Lemma 16.

Theorem 17. *Problem 2 can be solved in $O(n^6 \log n)$ time and $O(n^3)$ space.*

3.4 Solution to Problem 3

Consider Problem 3 of computing a substring that most frequently occurs in T.

Lemma 18. *For any non-empty strings T and P, $Occ(T, P[1:i]) \supseteq Occ(T,P)$ for any integer $0 \leq i \leq |P|$.*

The above monotonicity lemma implies that the empty string ε is always the solution for Problem 3. To make the problem more interesting, we consider the following version of the problem where the output is a substring of length at least 2.

Problem 5 (Computing most frequent substring of length at least 2 from SLP). Given an SLP $\mathcal{T}$ that derives a string T, compute a string P such that $|P| \geq 2$ and $|Occ(T,P)| \geq |Occ(T,Q)|$ for any other string Q with $|Q| \geq 2$.

Again, by the monotonicity lemma, it is sufficient only to consider a substring of length 2 as a solution to Problem 5.

The next lemma is fundamental for solving Problem 5.

Lemma 19. *For any SLP variables X_i and Y_j with $|Y_j| \geq 2$, $|Occ(X_i, Y_j)|$ can be computed in $O(n)$ time, with $O(mn^2)$-time $O(n^2)$-space preprocessing.*

Proof. Let D be a dynamic programming table of size $n \times n$ such that $D[i,j]$ represents how many times X_j appears in the derivation tree of X_i. After initializing all entries with 0, the value of each $D[i,j]$ is computed by the following recurrence:

$$D[i,j] = \begin{cases} 1 & \text{if } i = j, \\ D[\ell, j] + D[r, j] & \text{if } X_i = X_\ell X_r. \end{cases}$$

Then we obtain

$$|Occ(X_i, Y_j)| = \sum_{h=1}^{i} (D[i,h] \times |Occ^{\triangle}(X_h, Y_j) - \{|X_L| - |Y_j| + 1 \mid X_h = X_L X_R\}|).$$

We remove an occurrence of Y_j that touches the boundary of X_h from $Occ^{\triangle}(X_h, Y_j)$, since this occurrence covers the boundary of some other variable (recall we have assumed $|Y_j| \geq 2$).

In the preprocessing stage, we compute $Occ^{\triangle}(X_i, Y_j)$ for each $1 \leq i \leq n$ and $1 \leq j \leq m$. It takes $O(n^2m)$ time and $O(nm)$ space by Theorem 3. Then we compute the D-table in $O(n^2)$ time and space. Hence the preprocessing cost is $O(n^2m)$ time and $O(n^2)$ space, assuming $n \geq m$.

By Lemma 1, the value of $|Occ^{\triangle}(X_i, Y_m) - \{|X_\ell| - |Y_m| + 1\}|$ is computable in constant time. Thus we can compute $|Occ(T, P)|$ in $O(n)$ time and space. □

Theorem 20. *Problem 5 can be solved in $O(|\Sigma|^2n^2)$ time and $O(n^2)$ space.*

Proof. Let n be the size of SLP $\mathcal{T}$ and X_i denote its variable for $1 \leq i \leq n$. For each pair of variables $X_h = a$ and $X_j = b$ such that $a, b \in \Sigma$, we construct a new SLP $\mathcal{S}_{h,j} : Y_{h,j} = X_hX_j, X_h = a, X_j = b$. Then for each $\mathcal{S}_{h,j}$, we compute $Occ^{\triangle}(X_i, Y_{h,j})$ for every variable X_i of $\mathcal{T}$. Then a string $Y_{h,j}$ for which $|Occ(X_n, Y_{h,j})|$ is maximum is a solution to Problem 5.

Since the size of each new SLP $\mathcal{S}_{h,j}$ is constant, we can compute a DP table *App* that correspond to $\{Occ^{\triangle}(X_i, Y_{h,j})\}_{i=1}^{n}$ in $O(n^2)$ time and $O(n)$ space for each new SLP $\mathcal{S}_{h,j}$ by Theorem 3.

Due to Lemma 19, $|Occ(X_n, Y_{h,j})|$ can be computed in $O(n)$ time with $O(n^2)$ time and space preprocessing. Note that we can use the same D-table of Lemma 19 to compute $|Occ(X_n, Y_{h,j})|$ for every $Y_{h,j}$. On the other hand, we can discard the *App* table after $|Occ(X_n, Y_{h,j})|$ has been computed. Hence the total space requirement is $O(n^2)$. Since there are $O(|\Sigma|^2)$ new SLPs, it takes a total of $O(|\Sigma|^2n^2)$ time. □

3.5 Solution to Problem 4

Here we consider Problem 4 of computing a substring that has the most non-overlapping occurrences in a string T, when given a corresponding SLP $\mathcal{T}$.

For any string P to overlap itself, P has to be of length at least 2. Again, to make the problem more interesting, we consider the following problem.

Problem 6 (Computing most frequent non-overlapping substring of length at least 2 from SLP). Given an SLP $\mathcal{T}$ that derives a string T, compute a string P such that $|P| \geq 2$ and no other string Q with $|Q| \geq 2$ has more non-overlapping occurrences in T than P does.

The next lemma is a non-overlapping version of Lemma 18.

Lemma 21. *For any non-empty strings T and P, if there are two non-overlapping occurrences of P in T, then there are at least two non-overlapping occurrences of $P[1 : i]$ in T for any integer $0 \leq i \leq |P|$.*

Hence it suffices to consider a substring of length 2 as a solution to Problem 6.

We are now ready to show the following theorem.

Theorem 22. *Problem 6 can be solved in $O(n^4 \log n)$ time and $O(n^3)$ space.*

Proof. By Lemma 21, we consider a substring of length 2 as a solution to Problem 6.

For any string $P = ab$ with $a \neq b$, the set of its non-overlapping occurrences in T is identical to $Occ(T, P)$, since P cannot overlap with itself. Thus this case can be solved in the same way to Theorem 5.

Now consider any string $P = aa$. Although we will only show how to compute the number of its non-overlapping occurrences in T, it is easy to extend our method to computing a representation of its non-overlapping occurrences in T without increasing asymptotic complexities. For any $a \in \Sigma$ and any variable X_i, let $\alpha_{X_i,a} = LCPref(X_i, a^*)$ and $\beta_{X_i,a} = LCSuf(X_i, a^*)$ where a^* denotes an infinite repetition of a. Then, for any variable X_h, the number $H(X_h, aa)$ of non-overlapping occurrences of aa in X_h can be computed by the following recurrence:

$$H(X_h, aa) = \begin{cases} |Occ^{\triangle}(X_h, aa)| & \text{if } |X_h| \leq 2, \\ H(X_\ell, aa) + H(X_r, aa) & \\ -\lfloor \frac{\beta_{X_\ell,a}}{2} \rfloor + \lfloor \frac{\beta_{X_\ell,a}+\beta_{X_r,a}}{2} \rfloor - \lfloor \frac{\alpha_{X_r,a}}{2} \rfloor & \text{if } X_h = X_\ell X_r \text{ and } |X_h| > 2. \end{cases}$$

Consider the case where $|X_h| \leq 2$. For each variable X_h, $|Occ^{\triangle}(X_h, aa)|$ can be computed in total of $O(n^2)$ time and space, in the same way as mentioned in the proof of Theorem 20.

Now consider the other case. For any variable X_i, it holds that

$$\alpha_{X_i,a} = \begin{cases} 0 & \text{if } X_i[1] \neq a, \\ 1 + FM(X_i, X_i, 2) & \text{if } X_i[1] = a. \end{cases}$$

We can check whether $X_i[1] = a$ or not in $O(n)$ time, since the height of the derivation tree of X_i is at most $n + 1$. Therefore, we can compute $\alpha_{X_i,a}$ in $O(n \log n)$ time by Lemma 6. Similar arguments hold for computing $\beta_{X_i,a}$. The number of patterns of the form $P = aa$ is $O(|\Sigma|)$. Thus we need $O(|\Sigma| n \log n)$ time for this case.

To compute the FM function in $O(n \log n)$ time, we need to compute $OL(X_i, X_j)$ for any variables X_i and X_j, taking $O(n^4 \log n)$ time and $O(n^3)$ space due to Theorem 5. Overall, it takes $O(n^4 \log n)$ time and $O(n^3)$ space to solve Problem 6. □

4 Computing the Representative of a Given Pattern from Compressed Text

4.1 Problem

In this subsection, we begin with recalling the equivalence relations on strings introduced by Blumer et al. [1], and then state their properties.

We define two equivalence relations w.r.t. a string T based on Occ as follows. The equivalence relations $\equiv_L$ and $\equiv_R$ w.r.t. a string $T \in \Sigma^*$ are defined by:

$$Y \equiv_L Z \Leftrightarrow Occ(T, Y) = Occ(T, Z), \text{ and}$$
$$Y \equiv_R Z \Leftrightarrow Occ(T, Y) \oplus (|Y| - 1) = Occ(T, Z) \oplus (|Z| - 1),$$

where Y and Z are any strings in Σ^*. The equivalence classes of a string Y with respect to $\equiv_L$ and $\equiv_R$ are denoted by $[Y]_{\equiv_L}$ and $[Y]_{\equiv_R}$, respectively.

For any substring Y of T, let $\overrightarrow{Y}$ and $\overleftarrow{Y}$ denote the unique longest member of $[Y]_{\equiv_L}$ and $[Y]_{\equiv_R}$, respectively. Let $\overleftrightarrow{Y} = \alpha Y \beta$ such that $\alpha, \beta \in \Sigma^*$ are the strings satisfying $\overleftarrow{Y} = \alpha Y$ and $\overrightarrow{Y} = Y\beta$, respectively. Intuitively, $\overleftrightarrow{Y} = \alpha Y \beta$ means that:

– Every time Y occurs in T, it is preceded by α and followed by β.

– Strings α and β are longest possible.

We define another equivalence relation $\equiv$ w.r.t. T by $Y \equiv Z \Leftrightarrow \overleftrightarrow{Y} = \overleftrightarrow{Z}$, and the equivalence class of a string Y is denoted by $[Y]_{\equiv}$. String $\overleftrightarrow{Y}$ is called the *representative* of the equivalence class $[Y]_{\equiv}$.

The problem to be tackled in this section is the following.

Problem 7. Given two SLPs $\mathcal{T}$ and $\mathcal{P}$ that derive strings T and P respectively, compute an occurrence position and the length of the representative $\overleftrightarrow{P}$ w.r.t. T, if $|Occ(T, P)| \geq 1$.

4.2 Solution to Problem 7

Let m be the size of SLP $\mathcal{P}$, and let Y_j denote each variable of SLP $\mathcal{P}$ for $1 \leq j \leq m$. Assume without loss of generality that $m \leq n$.

Theorem 23. *Problem 7 can be solved in $O(n^4 \log n)$ time and $O(n^3)$ space.*

Proof. We only show how to compute the length and an occurrence position of $\overleftarrow{P}$, since those of $\overrightarrow{P}$ and $\overleftrightarrow{P}$ can be computed similarly.

We first compute $Occ^{\triangle}(X_i, Y_m)$ for each X_i according to Theorem 3.

If the length of $\overleftarrow{P}$ is known, then it is trivial that an occurrence position of $\overleftarrow{P}$ can be computed from an occurrence position of $P = Y_m$ in $T = X_n$. An occurrence position of Y_m in X_n can be retrieved in $O(n^2)$ time using the m-th column of the App table that correspond to $Occ^{\triangle}(X_1, Y_m), \ldots, Occ^{\triangle}(X_n, Y_m)$. Hence we concentrate on how to compute the length of $\overleftarrow{P}$ in the sequel.

For any variables X_i and X_h, and integers $1 \leq k_i \leq |X_i|$ and $1 \leq k_h \leq |X_h|$, let $LE_{X_i,X_h}(k_i.k_h)$ be the largest integer $p \geq 1$ such that $X_i[k_i - p : k_i - 1] = X_h[k_h - p : k_h - 1]$. If such p does not exist, then let $LE_{X_i,X_h}(k_i, k_h) = 0$.

Depending on the cardinality of $Occ^{\triangle}(X_i, Y_m)$, we have the following cases:

1. When $|Occ^{\triangle}(X_i, Y_m)| = 0$ for every variable X_i. Then clearly there is no answer to Problem 7.
2. When $|Occ^{\triangle}(X_i, Y_m)| = 1$ for some variable X_i and $|Occ^{\triangle}(X_{i'}, Y_m)| = 0$ for every $X_{i'} \neq X_i$. In this case, we have that $|Occ(T, P)| = 1$. Hence, by definition, we have $|\overleftarrow{P}| = k + |P| - 1$ where $\{k\} = Occ(T, P)$.
3. When $0 \leq |Occ^{\triangle}(X_i, Y_m)| \leq 1$ for any variable X_i and there are at least two variables such that $|Occ^{\triangle}(X_i, Y_m)| = 1$. For any variable X_i such that $|Occ^{\triangle}(X_i, Y_m)| = 1$, let $\{k_i\} = Occ^{\triangle}(X_i, Y_m)$. Let A and B be the sets of variable pairs such that

$$A = \{(X_i, X_h) \mid LE_{X_i,X_h}(k_i, k_h) < \min\{k_i, k_h\}\},$$
$$B = \{(X_i, X_h) \mid LE_{X_i,X_h}(k_i, k_h) = \min\{k_i, k_h\}\}.$$

See also Figure 5.

(a) When $\min\{LE_{X_i,X_h}(k_i, k_h) \mid (X_i, X_h) \in A\} \leq \min\{LE_{X_i,X_h}(k_i, k_h) \mid (X_i, X_h) \in B\}$. In this case, we have

$$|\overleftarrow{P}| = |\overleftarrow{Y_m}| = \min\{LE_{X_i,X_h}(k_i, k_h) \mid (X_i, X_h) \in A\} + |Y_m|.$$

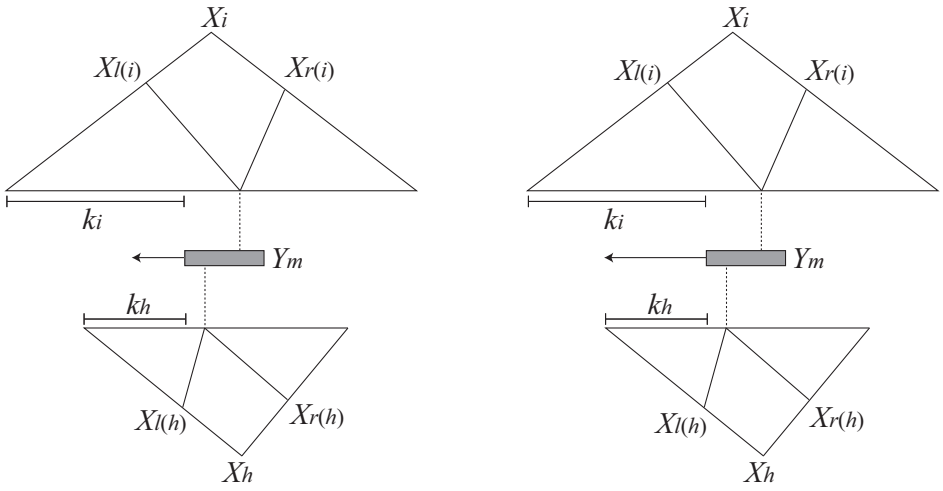

Figure 5. $LE_{X_i,X_h}(k_i,k_h) < \min\{k_i,k_h\}$ (left) and $LE_{X_i,X_h}(k_i,k_h) = \min\{k_i,k_h\}$ (right).

(b) When $\min\{LE_{X_i,X_h}(k_i,k_h) \mid (X_i,X_h) \in A\} > \min\{LE_{X_i,X_h}(k_i,k_h) \mid (X_i,X_h) \in B\}$.
For any pair of variables $X_i, X_h \in B$, assume w.l.o.g. that $k_i \geq k_h$. Let $X_h = X_{\ell(h)}X_{r(h)}$.
For any variable X_i such that $|Occ^{\triangle}(X_i,Y_m)| = 1$, we compute

$$G_i = Occ(X_i, X_{\ell(h)}) \cap (Occ^{\triangle}(X_i,Y_m) \ominus k_h).$$

Note that $|G_i| \leq 1$, since $|Occ^{\triangle}(X_i,Y_m)| = 1$. Let X_i, X_j be any variables such that $G_i = \{g_i\}$, $G_j = \{g_j\}$ and $g_i \leq g_j$. We compute $LE_{X_i,X_j}(g_i,g_j)$ until $LE_{X_i,X_j}(g_i,g_j) < \min\{g_i,g_j\}$ or X_i is a prefix of T (see Figure 6).

i. When $\min\{LE_{X_i,X_h}(Y_m,k_i,k_h) \mid (X_i,X_h) \in A\} \geq LE_{X_i,X_j}(g_i,g_j)$. In this case, $LE_{X_i,X_j}(g_i,g_j) + |Y_m|$ is a new candidate for $|\overleftarrow{Y_m}|$.
ii. When $\min\{LE_{X_i,X_h}(Y_m,k_i,k_h) \mid (X_i,X_h) \in A\} < LE_{X_i,X_j}(g_i,g_j)$. In this case, $LE_{X_i,X_j}(g_i,g_j) + |Y_m|$ is not a candidate for $|\overleftarrow{Y_m}|$.

4. When $0 \leq |Occ^{\triangle}(X_i,Y_m)| \leq 2$ for any $1 \leq i \leq n$. This case can be solved in a similar way to Case 3.
5. When $|Occ^{\triangle}(X_i,Y_m)| \geq 3$ for some $1 \leq i \leq n$. It follows from Lemma 1 that $Y_m = P = u^d v$ where $|u|$ is the smallest period of P, $d \geq 2$ is a positive integer, and v is a proper (possibly empty) prefix of u. It now holds that $|\overleftarrow{Y_m}| < |u| + |Y_m|$, since every occurrence of Y_m in $Occ^{\triangle}(X_i,Y_m)$ except for the first one is always preceded by u.
The length of $\overleftarrow{Y_m}$ can be computed as follows. See also Figure 7. For all variables $X_i = X_{\ell(i)}X_{r(i)}$ such that $|Occ^{\triangle}(X_i,Y_m)| \geq 1$, compute $FM(\overline{X_{\ell(i)}}, \overline{X_{\ell(i)}}, |u|+1)$. Then we have

$$|\overleftarrow{P}| = |\overleftarrow{Y_m}| = \min\{FM(\overline{X_{\ell(i)}}, \overline{X_{\ell(i)}}, |u|+1) \mod |u| \mid |Occ^{\triangle}(X_i,Y_m)| \geq 1\} + |Y_m|.$$

Now we analyze the complexity. By Theorem 3, $Occ^{\triangle}(X_i,Y_m)$ can be computed in $O(n^3)$ time with $O(n^2)$ space. Moreover, the cardinality, and a membership query to each $Occ^{\triangle}(X_i,Y_m)$ is answered in constant time due to Lemma 1. Therefore, Cases 1 and 2 can be solved in constant time provided that $Occ^{\triangle}(X_i,Y_m)$ are pre-computed.

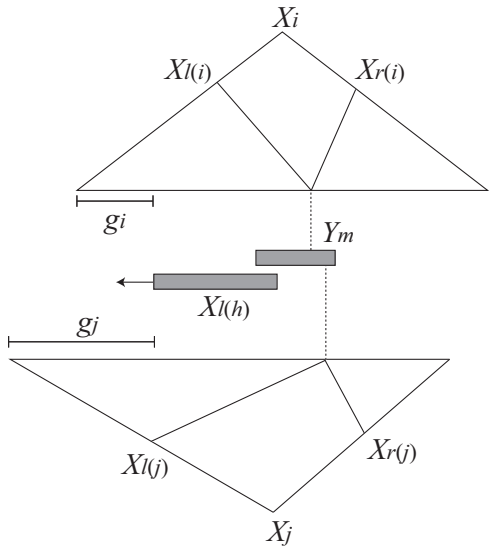

Figure 6. Compute $LE_{X_i,X_j}(g_i, g_j)$ until $LE_{X_i,X_j}(g_i, g_j) < \min\{g_i, g_j\}$ or X_i is a prefix of T.

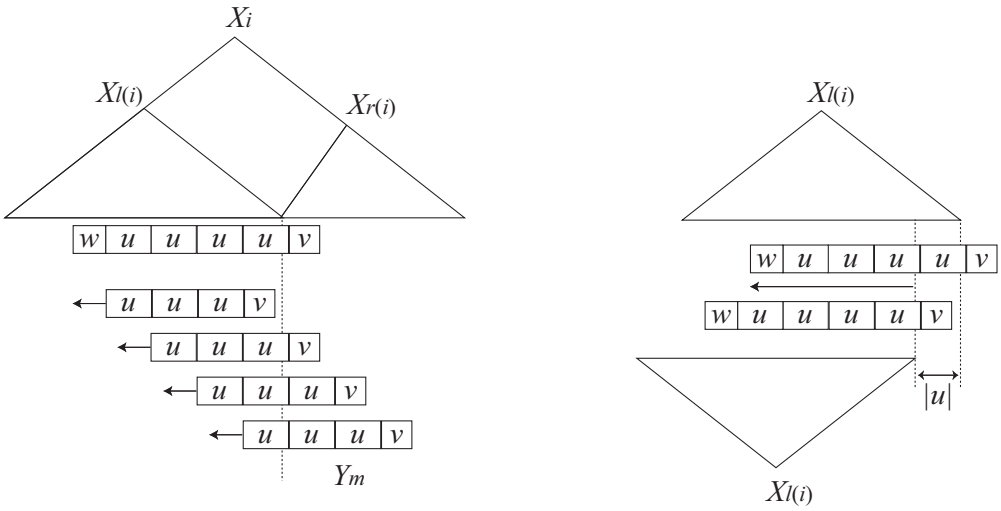

Figure 7. Illustration for Case 5. String w is a proper suffix of u such that every occurrence of $Y_m = u^d v$ is preceded by w (left). The length of w can be computed as $|w| = FM(\overline{X_{\ell(i)}}, \overline{X_{\ell(i)}}, |u| + 1) \bmod |u|$ (right).

Now consider Case 3. We compute the value of $LE_{X_i,X_h}(k_i, k_h)$ for all pairs of variables, whose number is $O(n^2)$. Since $\{k_i\} = Occ^{\triangle}(X_i, Y_m)$ and $\{k_h\} = Occ^{\triangle}(X_h, Y_m)$, it holds that

$$LE_{X_i,X_h}(k_i, k_h) = FM(\overline{X_i}, \overline{X_{\ell(h)}}, |X_i| - k_i - |X_{\ell(h)}| + k_j + 1) - |X_{\ell(h)}| + k_h. \quad (2)$$

It follows from Lemma 6 that Case 3a can be solved in $O(n^3 \log n)$ time. Now focus on Case 3b. For any variable X_i, G_i can be computed in $O(n)$ time, since $|Occ^{\triangle}(X_i, Y_m)| = 1$ and a membership query to $Occ(X_i, X_{\ell(h)})$ can be answered in $O(n)$ time due to Lemma 2. In each step of the loop, we compute the value of $LE_{X_i,X_j}(g_i, g_j) + |Y_m|$ for $O(n^2)$ pairs of variables. During this loop, the value of $LE_{X_i,X_j}(g_i, g_j) + |Y_m|$ is monotonically non-decreasing, and the size of G_i is mono-

tonically non-increasing. Hence, the depth of the loop is bounded by n. Moreover, we have that

$$LE_{X_i,X_j}(g_i, g_j) = FM(\overline{X_j}, \overline{X_{\ell(i)}}, |X_i| - k_j - |X_{\ell(i)}| + k_i + 1) - |X_{\ell(i)}| + g_i. \quad (3)$$

By Equation (3) and Lemma 6, the total time cost for Case 3b is $O(n^4 \log n)$. Therefore, Case 3 is solvable in $O(n^4 \log n)$ time, and so is Case 4.

In Case 5 we call the FM function at most n times, and each call of the FM function takes $O(n \log n)$ time by Lemma 6. Hence Case 5 can be solved in $O(n^2 \log n)$ time.

Computing $OL(X_i, X_j)$ for each pair of variables X_i, X_j requires $O(n^4 \log n)$ time and $O(n^3)$ space due to Theorem 5. Overall, we conclude that Problem 7 can be solved in $O(n^4 \log n)$ time with $O(n^3)$ space. □

References

1. A. Blumer, J. Blumer, D. Haussler, R. Mcconnell, and A. Ehrenfeucht: *Complete inverted files for efficient text retrieval and analysis.* J. ACM 34(3), 1987, pp. 578–595.
2. R. Feldman and J. Sanger: *The Text Mining Handbook: Advanced Approaches in Analyzing Unstructured Data*, Cambridge University Press, 2007.
3. L. Gąsieniec, M. Karpinski, W. Plandowski, and W. Rytter: *Efficient algorithms for Lempel-Ziv encoding*, in Proc. 5th Scandinavian Workshop on Algorithm Theory (SWAT'96), vol. 1097 of Lecture Notes in Computer Science, Springer-Verlag, 1996, pp. 392–403.
4. M. Karpinski, W. Rytter, and A. Shinohara: *An efficient pattern-matching algorithm for strings with short descriptions.* Nordic Journal of Computing, 4 1997, pp. 172–186.
5. J. Kieffer, E. Yang, G. Nelson, and P. Cosman: *Universal lossless compression via multilevel pattern matching.* IEEE Transactions on Information Theory, 46(4) 2000, pp. 1227–1245.
6. Y. Lifshits: *Solving classical string problems an compressed texts*, in Combinatorial and Algorithmic Foundations of Pattern and Association Discovery, no. 06201 in Dagstuhl Seminar Proceedings, 2006.
7. Y. Lifshits: *Processing compressed texts: A tractability border*, in Proc. 18th Annual Symposium on Combinatorial Pattern Matching (CPM'07), vol. 4580 of Lecture Notes in Computer Science, Springer-Verlag, 2007, pp. 228–240.
8. W. Matsubara, S. Inenaga, A. Ishino, A. Shinohara, T. Nakamura, and K. Hashimoto: *Efficient algorithms to compute compressed longest common substrings and compressed palindromes.* Theoretical Computer Science, 410(8–10) 2009, pp. 900–913.
9. M. Miyazaki, A. Shinohara, and M. Takeda: *An improved pattern matching algorithm for strings in terms of straight-line programs*, in Proc. 8th Annual Symposium on Combinatorial Pattern Matching (CPM'97), vol. 1264 of Lecture Notes in Computer Science, Springer-Verlag, 1997, pp. 1–11.
10. K. Narisawa, H. Bannai, K. Hatano, and M. Takeda: *Unsupervised spam detection based on string alienness measures*, in Proc. 10th International Conference on Discovery Science (DS'07), vol. 4755 of Lecture Notes in Artificial Intelligence, 2007, pp. 161–172.
11. C. G. Nevill-Manning, I. H. Witten, and D. L. Maulsby: *Compression by induction of hierarchical grammars*, in Proc. Data Compression Conference '94 (DCC'94), IEEE Computer Society, 1994, pp. 244–253.
12. W. Rytter: *Grammar compression, LZ-encodings, and string algorithms with implicit input*, in Proc. 31st International Colloquium on Automata, Languages and Programming (ICALP'04), vol. 3142 of Lecture Notes in Computer Science, Springer-Verlag, 2004, pp. 15–27.
13. M. Takeda, T. Matsumoto, T. Fukuda, and I. Nanri: *Discovering characteristic expressions in literary works.* Theoretical Computer Science, 292(2) 2003, pp. 525–546.
14. J. Ziv and A. Lempel: *A universal algorithm for sequential data compression.* IEEE Transactions on Information Theory, IT-23(3) 1977, pp. 337–349.
15. J. Ziv and A. Lempel: *Compression of individual sequences via variable-length coding.* IEEE Transactions on Information Theory, 24(5) 1978, pp. 530–536.