Computing longest common substring and all palindromes from compressed strings

Wataru Matsubara[1], Shunsuke Inenaga[2], Akira Ishino[1], Ayumi Shinohara[1],
Tomoyuki Nakamura[1], and Kazuo Hashimoto[1]

[1] Graduate School of Information Science, Tohoku University, Japan
{matsubara@shino., ishino@, ayumi@, nakamura@aiet.,
hk@aiet.}ecei.tohoku.ac.jp
[2] Department of Computer Science and Communication Engineering,
Kyushu University, Japan
inenaga@c.csce.kyushu-u.ac.jp

Abstract. This paper studies two problems on compressed strings described in terms of *straight line programs* (*SLPs*). One is to compute the length of the longest common substring of two given SLP-compressed strings, and the other is to compute all palindromes of a given SLP-compressed string. In order to solve these problems efficiently (in polynomial time w.r.t. the compressed size) decompression is never feasible, since the decompressed size can be exponentially large. We develop combinatorial algorithms that solve these problems in $O(n^4 \log n)$ time with $O(n^3)$ space, and in $O(n^4)$ time with $O(n^2)$ space, respectively, where n is the size of the input SLP-compressed strings.

1 Introduction

The importance of algorithms for *compressed texts* has recently been arising due to the massive increase of data that are treated in compressed form. Of various text compression schemes introduced so far, *straight line program* (*SLP*) is one of the most powerful and general compression schemes. An SLP is a context-free grammar of either of the forms $X \to YZ$ or $X \to a$, where a is a constant. SLP allows *exponential* compression, i.e., the original (uncompressed) string length N can be exponentially large w.r.t. the corresponding SLP size n. In addition, resulting encoding of most grammar- and dictionary-based text compression methods such as LZ-family [1, 2], run-length encoding, multi-level pattern matching code [3], Sequitur [4] and so on, can quickly be transformed into SLPs [5–7]. Therefore, it is of great interest to analyze what kind of problems on SLP-compressed strings can be solved in polynomial time w.r.t. n. Moreover, for those that are polynomial solvable, it is of great importance to design efficient algorithms. In so doing, one has to notice that decompression is never feasible, since it can require exponential time and space w.r.t. n.

The first polynomial time algorithm for SLP-compressed strings was given by Plandowski [8], which tests the equality of two SLP-compressed strings in $O(n^4)$ time. Later on Karpinski et al. [9] presented an $O(n^4 \log n)$-time algorithm for the substring pattern matching problem for two SLP-compressed strings. Then it was improved to $O(n^4)$ time by Miyazaki et al. [10] and recently to $O(n^3)$ time by Lifshits [11]. The problem of computing the minimum period of a given SLP-compressed string was shown to be solvable in $O(n^4 \log n)$ time [9], and lately in $O(n^3 \log N)$ time [11]. Gąsieniec et al. [5] claimed that all squares of a given SLP-compressed string can be computed in $O(n^6 \log^5 N)$ time.

On the other hand, there are some hardness results on SLP-compressed string processing. Lifshits and Lohrey [12] showed that the subsequence pattern matching problem for SLP-compressed strings is NP-hard, and that computing the length of the longest common subsequence of two SLP-compressed strings is also NP-hard. Lifshits [11] showed that computing the Hamming distance between two SLP-compressed strings is #P-complete.

In this paper we tackle the following two problems: one is to compute the length of the *longest common substring* of two SLP-compressed strings, and the other is to find all maximal *palindromes* of an SLP-compressed string. The first problem is listed as an open problem in [11]. This paper closes the problem giving an algorithm that runs in $O(n^4 \log n)$ time with $O(n^3)$ space. For second the problem of computing all maximal palindromes, we give an algorithm that runs in $O(n^4)$ time with $O(n^2)$ space.

Comparison to previous work. *Composition system* is a generalization of SLP which also allows "truncations" for the production rules. Namely, a rule of composition systems is of one of the following forms: $X \to Y^{[i]} Z_{[j]}$, $X \to YZ$, or $X \to a$, where $Y^{[i]}$ and $Z_{[j]}$ denote the prefix of length i of Y and the suffix of length j of Z, respectively. Gąsieniec et al. [5] presented an algorithm that computes all maximal palindromes from a given composition system in $O(n \log^2 N \times Eq(n))$ time, where $Eq(n)$ denotes the time needed for the equality test of composition systems. Since $Eq(n) = O(n^4 \log^2 N)$ in [5], the overall time cost is $O(n^5 \log^4 N)$.

Limited to SLPs, $Eq(n) = O(n^3)$ due to the recent work by Lifshits [11]. Still, computing all maximal palindromes takes $O(n^4 \log^2 N)$ time in total, and therefore our solution with $O(n^4)$ time is faster than the previous known ones (recall that $N = O(2^n)$). The space requirement of the algorithm by Gąsieniec et al. [5] is unclear. However, since the equality test algorithm of [11] takes $O(n^2)$ space, the above-mentioned $O(n^4 \log^2 N)$-time solution takes at least as much space as ours.

2 Preliminaries

For any set U of pairs of integers, we denote $U \oplus k = \{(i+k, j+k) \mid (i,j) \in U\}$. We denote by $\langle a, d, t \rangle$ the arithmetic progression with the minimal element a, the common difference d and the number of elements t, that is, $\langle a, d, t \rangle = \{a + (i-1)d \mid 1 \leq i \leq t\}$. When $t = 0$, let $\langle a, d, t \rangle = \emptyset$.

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 any string T, let T^R denote the reversed string of T, namely, $T^R = T[|T|] \cdots T[2]T[1]$.

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

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

A non-empty string T such that $T = T^R$ is said to be a *palindrome*. When $|T|$ is even, then T is said to be an *even palindrome*, that is, $T = SS^R$ for some $S \in \Sigma^+$. Similarly, when $|T|$ is odd, then T is said to be an *odd palindrome*, that is, $T = ScS^R$ for some $S \in \Sigma^*$ and $c \in \Sigma$. For any string T and its substring $T[i:j]$ such that $T[i:j] = T[i:j]^R$, $T[i:j]$ is said to be the *maximal palindrome* w.r.t. the center $\lfloor \frac{i+j}{2} \rfloor$, if either $T[i-1] \neq T[j+1]$, $i = 1$, or $j = |T|$. In particular, $T[1:j]$ is said to be a *prefix palindrome* of T, and $T[i:|T|]$ is said to be a *suffix palindrome* of T.

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 in either of the following form:

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

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 $\mathcal{T}$.

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 X_i^R as follows:

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

Let $\mathcal{T}^R$ be the SLP consisting of variables X_i^R for $1 \leq i \leq n$.

Lemma 1. *SLP $\mathcal{T}^R$ derives string T^R.*

Proof. By induction on the variables X_i^R. Let Σ_T be the set of characters appearing in T. For any $1 \leq i \leq |\Sigma_T|$, we have $X_i = a$ for some $a \in \Sigma_T$, thus $X_i^R = a$ and $a = a^R$. Let T_i denote the string derived from X_i. For the induction hypothesis, assume that X_j^R derives T_j^R for any $1 \leq j \leq i$. Now consider variable $X_{i+1} = X_\ell X_r$. Note $T_{i+1} = T_\ell T_r$, which implies $T_{i+1}^R = T_r^R T_\ell^R$. By definition, we have $X_{i+1}^R = X_r^R X_\ell^R$. Since $\ell, r < i+1$, by the induction hypothesis X_{i+1}^R derives $T_r^R T_\ell^R = T_{i+1}^R$. Thus, $\mathcal{T}^R = X_n^R$ derives $T_n^R = T^R$. □

Note that $\mathcal{T}^R$ can be easily computed from $\mathcal{T}$ in $O(n)$ time.

3 Computing Longest Common Substring of Two SLP Compressed Strings

Let $\mathcal{T}$ and $\mathcal{S}$ be the SLPs of sizes n and m, which describe strings T and S, respectively. Without loss of generality we assume that $n \geq m$.

In this section we tackle the following problem:

Problem 1. Given two SLPs $\mathcal{T}$ and $\mathcal{S}$, compute $LCStr(T, S)$.

In what follows we present an algorithm that solves Problem 1 in $O(n^4 \log n)$ time and $O(n^3)$ space. Let X_i and Y_j denote any variable of $\mathcal{T}$ and $\mathcal{S}$ for $1 \leq i \leq n$ and $1 \leq j \leq m$.

3.1 Overlaps between Two Strings

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

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

Namely, $OL(X, Y)$ is the set of lengths of overlaps of suffixes of X and prefixes of Y. Karpinski et al. [9] gave the following results for computation of OL for strings described by SLPs.

Lemma 2 ([9]). *For any variables X_i and X_j of an SLP $\mathcal{T}$, $OL(X_i, X_j)$ can be represented by $O(n)$ arithmetic progressions.*

Theorem 1 ([9]). *For any SLP $\mathcal{T}$, $OL(X_i, X_j)$ can be computed in total of $O(n^4 \log n)$ time and $O(n^3)$ space for any $1 \leq i \leq n$ and $1 \leq j \leq n$.*

As we will show in the sequel, we need to compute $OL(X_i, Y_j)$ and $OL(Y_j, X_i)$ for any $1 \leq i \leq n$ and $1 \leq j \leq m$. In so doing, we produce a new variable $V = X_n Y_m$, that is, V is a concatenation of SLPs $\mathcal{T}$ and $\mathcal{S}$. Then we compute OL for each pair of variables in the new SLP of size $n + m$. On the assumption that $n \geq m$, it takes $O(n^4 \log n)$ time and $O(n^3)$ space in total.

3.2 The *FM* function

For any two variables X_i, Y_j and integer k with $1 \leq k \leq |X_i|$, we define the function $FM(X_i, Y_j, k)$ which returns the previous position of the first position of mismatches, when we compare Y_j with X_i at position k. Formally,

$$FM(X_i, Y_j, k) = \min\{1 \leq h \leq |Y_j| \mid X_i[k + h - 1] \neq Y_j[h]\} - 1.$$

Namely, $FM(X_i, Y_j, k)$ equals the length of the common prefix of $X_i[k : |X_i|]$ and Y_j when it is not zero. When the common prefix is the empty string ε (when such h does not exist), let $FM(X_i, Y_j, k) = 0$.

Lemma 3 ([9]). *For any 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 $OL(X_{i'}, X_{j'})$ is already computed for any $1 \leq i' \leq i$ and $1 \leq j' \leq j$.*

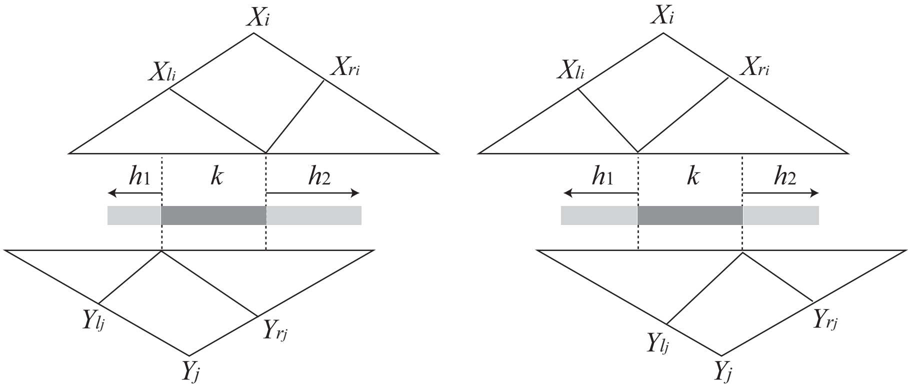

Fig. 1. Illustration of Observation 2 where we "extend" an overlap k as a candidate of $LCStr(T, S)$.

3.3 Efficient Computation of Longest Common Substrings

The main idea of our algorithm for computing $LCStr(T, S)$ is based on the following observation.

Observation 1 *For any substring Z of string T, there always exists a variable $X_i = X_{\ell_i} X_{r_i}$ of SLP $\mathcal{T}$ such that:*

- *Z is a substring of X_i and*
- *Z touches or covers the boundary between X_{ℓ_i} and X_{r_i}.*

It directly follows from the above observation that any common substring of strings T, S touches or covers both of the boundaries in X_i and Y_j for some $1 \leq i \leq n$ and $1 \leq j \leq m$.

For any SLP variables $X_i = X_{\ell_i} X_{r_i}$ and $Y_j = Y_{\ell_j} Y_{r_j}$, and $k \in OL(X_i, Y_j)$, let $Ext_{X_i,Y_j}(k) = k + h_1 + h_2$ such that $h_1 = LCSuf(X_{\ell_i}[1 : |X_{\ell_i}| - k], Y_{\ell_j})$ and $h_2 = LCPref(X_{r_i}, Y_{r_j}[k+1 : |Y_{r_j}|])$. For any $k \notin OL(X_i, Y_j)$, we leave $Ext_{X_i,Y_j}(k)$ undefined. For a set S of integers, we define $Ext_{X_i,Y_j}(S) = \{Ext_{X_i,Y_j}(k) \mid k \in S\}$. $Ext_{Y_j,X_i}(k)$ and $Ext_{Y_j,X_i}(S)$ are defined similarly.

The next observation follows from the above arguments (see also Fig. 1):

Observation 2 *For any strings T and S, $LCStr(T, S)$ equals to the maximum element of the set*

$$\bigcup_{1 \leq i \leq n, 1 \leq j \leq m} (Ext_{X_i,Y_j}(OL(X_{\ell_i}, Y_{r_j})) \cup Ext_{Y_j,X_i}(OL(Y_{\ell_j}, X_{r_i})) \cup LCStr^*(X_i, Y_j)),$$

$$\textit{where} \quad LCStr^*(X_i, Y_j) = LCSuf(X_{\ell_i}, Y_{\ell_j}) + LCPref(X_{r_i}, Y_{r_j}).$$

Based on Observation 2, our strategy for computing $LCStr(T, S)$ is to compute $\max(Ext_{X_i,Y_j}(OL(X_{\ell_i}, Y_{r_j})))$ and $\max(Ext_{Y_j,X_i}(OL(Y_{\ell_j}, X_{r_i})))$ for each pair of X_i and Y_j. Lemma 4 shows how to compute $\max(Ext_{X_i,Y_j}(OL(X_{\ell_i}, Y_{r_j})))$ and $\max(Ext_{Y_j,X_i}(OL(Y_{\ell_j}, X_{r_i})))$ using FM.

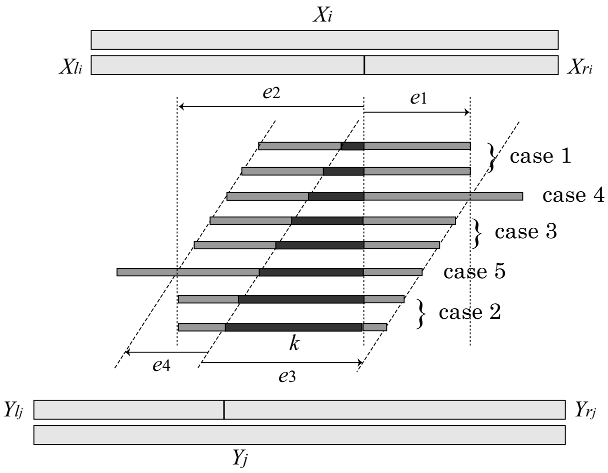

Fig. 2. Illustration for the proof of Lemma 4. The dark rectangles represent the overlaps between X_{ℓ_i} and Y_{r_j}. Case 6 is the special case where cases 4 and 5 happen at the same time and case 3 does not exist.

Lemma 4. *For any variables* $X_i = X_{\ell_i} X_{r_i}$ *and* $Y_j = Y_{\ell_j} Y_{r_j}$, *we can compute* $\max(Ext_{X_i,Y_j}(OL(X_{\ell_i}, Y_{r_j})))$ *and* $\max(Ext_{Y_j,X_i}(OL(Y_{\ell_j}, X_{r_i})))$ *in* $O(n^2 \log n)$ *time.*

Proof. Here we concentrate on computing $\max(Ext_{X_i,Y_j}(OL(X_{\ell_i}, Y_{r_j})))$, as the case of $\max(Ext_{Y_j,X_i}(OL(Y_{\ell_j}, X_{r_i})))$ is just symmetric. Let $\langle a, d, t \rangle$ be any of the $O(n)$ arithmetic progressions of $OL(X_{\ell_i}, Y_{r_j})$.

Assume that $t > 1$ and $a < d$. The cases where $t = 1$ or $a = d$ are easier to show. Let $u = Y_{r_j}[1 : a]$ and $v = Y_{r_j}[a+1 : d]$. For any string w, let w^* denote an infinite repetition of w, that is, $w^* = www\cdots$. Firstly we compute

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

$$e_2 = LCSuf(X_{\ell_i}, (vu)^*) = FM(X^R_{\ell_i}, X^R_{\ell_i}, d+1) + d,$$

$$e_3 = LCPref(Y_{r_j}, (uv)^*) = FM(Y_{r_j}, Y_{r_j}, d+1) + d,$$

$$e_4 = LCSuf(Y_{\ell_j}, (uv)^*) = \begin{cases} FM(X^R_{\ell_i}, Y^R_{\ell_j}, a{+}1) & \text{if } FM(X^R_{\ell_i}, Y^R_{\ell_j}, a{+}1) < d, \\ FM(Y^R_{\ell_j}, Y^R_{\ell_j}, d+1) + d & \text{otherwise.} \end{cases}$$

(See also Fig. 2.) As above, we can compute e_1, e_2, e_3, e_4 by at most 6 calls of FM. Note that $X_i[|X_{\ell_i}| - e_2 + 1 : |X_{\ell_i}| + e_1]$ is the longest substring of X_i that contains $X_i[|X_{\ell_i}| - d + 1 : |X_{\ell_i}|]$ and has a period d. Note also that $Y_j[|Y_{\ell_j}| - e_4 + 1 : |Y_{\ell_j}| + e_3]$ is the longest substring of Y_j that contains $Y_j[|Y_{\ell_j}| + 1 : |Y_{\ell_j}| + d]$ and has a period d.

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

case 1: When $k < \min\{e_3 - e_1, e_2 - e_4\}$. If $k - d \in \langle a, d, t\rangle$, it is not difficult to see $\mathit{Ext}_{X_i,Y_j}(k) = \mathit{Ext}_{X_i,Y_j}(k-d) + d$. Therefore, we have

$$A = \max\{\mathit{Ext}_{X_i,Y_j}(k) \mid k < \min\{e_3 - e_1, e_2 - e_4\}\} = \mathit{Ext}_{X_i,Y_j}(k'),$$

where $k' = \max\{k \mid k < \min\{e_3 - e_1, e_2 - e_4\}\}$.

case 2: When $k > \max\{e_3 - e_1, e_2 - e_4\}$. If $k + d \in \langle a, d, t\rangle$, it is not difficult to see $\mathit{Ext}_{X_i,Y_j}(k) = \mathit{Ext}_{X_i,Y_j}(k+d) + d$. Therefore, we have

$$B = \max\{\mathit{Ext}_{X_i,Y_j}(k) \mid k > \max\{e_3 - e_1, e_2 - e_4\}\} = \mathit{Ext}_{X_i,Y_j}(k''),$$

where $k'' = \min\{k \mid k > \max\{e_3 - e_1, e_2 - e_4\}\}$.

case 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 $\mathit{Ext}_{X_i,Y_j}(k) = \min\{e_1 + e_2, e_3 + e_4\}$ for any k with $\min\{e_3 - e_1, e_2 - e_4\} < k < \max\{e_3 - e_1, e_2 - e_4\}$. Thus

$$\begin{aligned} C &= \max\{\mathit{Ext}_{X_i,Y_j}(k) \mid \min\{e_3 - e_1, e_2 - e_4\} < k < \max\{e_3 - e_1, e_2 - e_4\}\} \\ &= \min\{e_1 + e_2, e_3 + e_4\}. \end{aligned}$$

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

$$\begin{aligned} D = \mathit{Ext}_{X_i,Y_j}(k) &= k + \min\{e_2 - k, e_4\} + \mathit{LCPref}(Y_{r_j}[k+1 : |Y_{r_j}|], X_{r_i}) \\ &= k + \min\{e_2 - k, e_4\} + \mathit{FM}(Y_{r_j}, X_{r_i}, k+1). \end{aligned}$$

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

$$\begin{aligned} E = \mathit{Ext}_{X_i,Y_j}(k) &= k + \mathit{LCSuf}(X_{\ell_i}[1 : |X_{\ell_i}| - k], Y_{\ell_j}) + \min\{e_1, e_3 - k\} \\ &= k + \mathit{FM}(X^R_{\ell_i}, Y^R_{\ell_j}, k+1) + \min\{e_1, e_3 - k\}. \end{aligned}$$

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

$$\begin{aligned} F &= \mathit{Ext}_{X_i,Y_j}(k) \\ &= k + \mathit{LCSuf}(X_{\ell_i}[1 : |X_{\ell_i}| - k], Y_{\ell_j}) + \mathit{LCPref}(Y_{r_j}[k+1 : |Y_{r_j}|], X_{r_i}) \\ &= k + \mathit{FM}(X^R_{\ell_i}, Y^R_{\ell_j}, k+1) + \mathit{FM}(Y_{r_j}, X_{r_i}, k+1). \end{aligned}$$

Then clearly the following inequality stands (see also Fig. 2):

$$F \geq \max\{D, E\} \geq C \geq \max\{A, B\}. \tag{1}$$

A membership query to the arithmetic progression $\langle a, d, t\rangle$ can be answered in constant time. Also, an element $k \in \langle a, d, t\rangle$ such that $\min\{e_3 - e_1, e_2 - e_4\} < k < \max\{e_3 - e_1, e_2 - e_4\}$ of case 3 can be found in constant time, if such exists. k' and k'' of case 1 and case 2, respectively, can be computed in constant time as well. Therefore, based on inequality (1), we can compute $\max(\mathit{Ext}_{X_i,Y_j}(\langle a, d, t\rangle))$ by at most 2 calls of FM, provided that e_1, e_2, e_3, e_4 are already computed.

Since $\mathit{OL}(X_{\ell_i}, Y_{r_j})$ contains $O(n)$ arithmetic progressions by Lemma 2, and each call of FM takes $O(n \log n)$ time by Lemma 3, $\max(\mathit{Ext}_{X_i,Y_j}(\mathit{OL}(X_{\ell_i}, Y_{r_j})))$ can be computed in $O(n^2 \log n)$ time. □

Now we obtain the main result of this section.

Theorem 2. *Problem 1 can be solved in $O(n^4 \log n)$ time with $O(n^3)$ space.*

Proof. It follows from Theorem 1 that $OL(X_i, Y_j)$ can be computed in $O(n^4 \log n)$ time with $O(n^3)$ space. For any variables $X_i = X_{\ell_i} X_{r_i}$ and $Y_j = Y_{\ell_j} Y_{r_j}$, by Lemmas 2, 3 and 4, $\max(Ext_{X_i,Y_j}(OL(X_{\ell_i}, Y_{r_j})))$ and $\max(Ext_{Y_j,X_i}(OL(Y_{\ell_j}, X_{r_i})))$ can be computed in $O(n^2 \log n)$ time.

Moreover, it is easy to see that

$$\begin{aligned} LCSuf(X_{\ell_i}, Y_{\ell_j}) &= FM(X_{\ell_i}^R, Y_{\ell_j}^R, 1) \text{ and} \\ LCPref(X_{r_i}, Y_{r_j}) &= FM(X_{r_i}, Y_{r_j}, 1). \end{aligned}$$

Thus $LCStr^*(X_i, Y_j)$ can be computed in $O(n \log n)$ time. Overall, by Observation 2 it takes $O(n^4 \log n)$ time and $O(n^3)$ to solve Problem 1. □

The following corollary is immediate.

Corollary 1. *Given two SLPs $\mathcal{T}$ and $\mathcal{S}$ describing strings T and S respectively, the beginning and ending positions of a longest common substring of T and S can be computed in $O(n^4 \log n)$ time with $O(n^3)$ space.*

4 Computing Palindromes from SLP Compressed Strings

In this section we present an efficient algorithm that computes a succinct representation of all maximal palindromes of string T, when its corresponding SLP $\mathcal{T}$ is given as input. The algorithm runs in $O(n^4)$ time and $O(n^2)$ space, where n is the size of the input SLP $\mathcal{T}$.

For any string T, let $Pals(T)$ denote the set of pairs of the beginning and ending positions of all maximal palindromes in T, namely,

$$Pals(T) = \{(p, q) \mid T[p : q] \text{ is the maximal palindrome centered at } \lfloor \tfrac{p+q}{2} \rfloor\}.$$

Note that the size of $Pals(T)$ is $O(|T|) = O(2^n)$. Thus we introduce a succinct representation of $Pals(T)$ in the next subsection.

4.1 Succinct Representation of *Pals*(*T*)

Let X_i denote a variable in $\mathcal{T}$ for $1 \leq i \leq n$. For any variables $X_i = X_\ell X_r$, let $Pals^\triangle(X_i)$ be the set of pairs of beginning and ending positions of maximal palindromes of X_i that cover or touch the boundary between X_ℓ and X_r, namely,

$$Pals^\triangle(X_i) = \{(p, q) \in Pals(X_i) \mid 1 \leq p \leq |X_\ell| + 1, |X_\ell| \leq q \leq |X_i|, p \leq q\}.$$

Also, let $PPals(T)$ and $SPals(T)$ denote the set of pairs of the beginning and ending positions of the prefix and suffix palindromes of T, respectively, that is,

$$\begin{aligned} PPals(T) &= \{(1, q) \in Pals(T) \mid 1 \leq q \leq |T|\}, \text{ and} \\ SPals(T) &= \{(p, |T|) \in Pals(T) \mid 1 \leq p \leq |T|\}. \end{aligned}$$

Gąsieniec et al. [5] claimed the following lemma:

Lemma 5 ([5]). *For any string T, $\mathit{PPals}(T)$ and $\mathit{SPals}(T)$ can be represented by $O(\log |T|)$ arithmetic progressions.*

We have the following observation for decomposition of $\mathit{Pals}(X_i)$.

Observation 3 *For any variables $X_i = X_\ell X_r$,*

$$\mathit{Pals}(X_i) = (\mathit{Pals}(X_\ell) - \mathit{SPals}(X_\ell)) \cup \mathit{Pals}^{\triangle}(X_i) \cup ((\mathit{Pals}(X_r) - \mathit{PPals}(X_r)) \oplus |X_\ell|).$$

Thus, the desired output $\mathit{Pals}(T) = \mathit{Pals}(X_n)$ can be represented as a combination of $\{\mathit{Pals}^{\triangle}(X_i)\}_{i=1}^n$, $\{\mathit{PPals}(X_i)\}_{i=1}^n$ and $\{\mathit{SPals}(X_i)\}_{i=1}^n$. Therefore, computing $\mathit{Pals}(T)$ is reduced to computing $\mathit{Pals}^{\triangle}(X_i)$, $\mathit{PPals}(X_i)$ and $\mathit{SPals}(X_i)$, for every $i = 1, 2, \ldots, n$. The problem to be tackled in this section follows:

Problem 2. Given an SLP $\mathcal{T}$ of size n, compute $\{\mathit{Pals}^{\triangle}(X_i)\}_{i=1}^n$, $\{\mathit{PPals}(X_i)\}_{i=1}^n$ and $\{\mathit{SPals}(X_i)\}_{i=1}^n$.

Lemma 6 is useful to compute $\mathit{Pals}^{\triangle}(X_i)$ from $\mathit{SPals}(X_\ell)$ and $\mathit{PPals}(X_r)$.

Lemma 6. *For any variable $X_i = X_\ell X_r$ and any $(p, q) \in \mathit{Pals}^{\triangle}(X_i)$, there exists an integer $l \geq 0$ such that $(p+l, q-l) \in \mathit{SPals}(X_\ell) \cup (\mathit{PPals}(X_r) \oplus |X_\ell|) \cup \{(|X_\ell|, |X_\ell|+1)\}$.*

Proof. Since $X_i[p : q]$ is a palindrome, $X_i[p + l : q - l]$ is also a palindrome for any $0 \leq l < \lfloor \frac{p+q}{2} \rfloor$. Then we have the following three cases:

1. When $\lfloor \frac{p+q}{2} \rfloor < |X_\ell|$, for $l = p - |X_\ell|$, we have $(p + l, q - l) \in \mathit{SPals}(X_\ell)$.
2. When $\lfloor \frac{p+q}{2} \rfloor > |X_\ell|$, for $l = |X_\ell| - p + 1$, we have $(p + l, q - l) \in \mathit{PPals}(X_r)$.
3. When $\lfloor \frac{p+q}{2} \rfloor = |X_\ell|$, if $q - p + 1$ is odd, then the same arguments to case 1 apply, since $X_\ell[|X_\ell|] = X_\ell[|X_\ell|]^R$ and $(|X_\ell|, |X_\ell|) \in \mathit{SPals}(X_\ell)$. If $q - p + 1$ is even, let $l = |X_\ell| - p$. In this case, we have $p + q = 2|X_\ell| + 1$. Thus, $p + l = |X_\ell|$ and $q - l = |X_\ell| + 1$.

□

By Lemma 6, $\mathit{Pals}^{\triangle}(X_i)$ can be computed by "extending" all palindromes in $\mathit{SPals}(X_\ell)$ and $\mathit{PPals}(X_r)$ to the maximal within X_i, and finding the maximal even palindromes centered at $|X_\ell|$ in X_i. In so doing, for any (maximal or non-maximal) palindrome $P = X_i[p : q]$, we define function Ext_{X_i} so that $\mathit{Ext}_{X_i}(p, q) = (p - h, q + h)$, where $h \geq 0$ and $X_i[p - h : q + h]$ is the maximal palindrome centered at position $\lfloor \frac{p+q}{2} \rfloor$ in X_i. For any p, q with $X_i[p : q]$ not being a palindrome, we leave $\mathit{Ext}_{X_i}(p, q)$ undefined. For a set S of pair of integers, let $\mathit{Ext}_{X_i}(S) = \{\mathit{Ext}_{X_i}(p, q) \mid (p, q) \in S\}$.

The next observations give us a recursive procedure to compute $\mathit{Pals}^{\triangle}(X_i)$.

Observation 4 *For any variable $X_i = X_\ell X_r$,*

$$\mathit{Pals}^{\triangle}(X_i) = \mathit{Ext}_{X_i}(\mathit{SPals}(X_\ell)) \cup \mathit{Ext}_{X_i}(\mathit{PPals}(X_r)) \cup \mathit{Pals}^*(X_i), \text{ where}$$

$$\mathit{Pals}^*(X_i) = \{(|X_\ell| - l + 1, |X_\ell| + l) \in \mathit{Pals}(X_i) \mid l \geq 1\}.$$

$\mathit{PPals}(X_i)$ and $\mathit{SPals}(X_i)$ can be computed from $\mathit{Pals}^{\triangle}(X_i)$ as follows:

Observation 5 *For any variable $X_i = X_\ell X_r$,*

$$\mathit{PPals}(X_i) = \mathit{PPals}(X_\ell) \cup \{(1, q) \in \mathit{Pals}^{\triangle}(X_i)\} \text{ and}$$
$$\mathit{SPals}(X_i) = (\mathit{SPals}(X_r) \oplus |X_\ell|) \cup \{(p, |X_i|) \in \mathit{Pals}^{\triangle}(X_i)\}.$$

4.2 Efficient Computation of $\mathit{Pals}^{\triangle}(X_i)$

Let us first briefly recall the work of [10, 11]. For any variables $X_i = X_\ell X_r$ and X_j, we define the set $\mathit{Occ}^{\triangle}(X_i, X_j)$ of all occurrences of X_j that cover or touch the boundary between X_ℓ and X_r, namely,

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

Theorem 3 ([11]). *For any variables X_i and X_j, $\mathit{Occ}^{\triangle}(X_i, X_j)$ can be computed in total of $O(n^3)$ time and $O(n^2)$ space.*

Lemma 7 ([10]). *For any variables X_i, X_j and integer k, $\mathit{FM}(X_i, X_j, k)$ can be computed in $O(n^2)$ time, provided that $\mathit{Occ}^{\triangle}(X_{i'}, X_{j'})$ is already computed for any $1 \le i' \le i$ and $1 \le j' \le j$.*

Lemma 8. *For any variable $X_i = X_\ell X_r$ and any arithmetic progression $\langle a, d, t\rangle$ with $(1, \langle a, d, t\rangle) \subseteq \mathit{PPals}(X_r)$, $\mathit{Ext}_{X_i}((1, \langle a, d, t\rangle))$ can be represented by at most 2 arithmetic progressions and a pair of the beginning and ending positions of a maximal palindrome, and can be computed by at most 4 calls of FM. Similar for $\mathit{Ext}_{X_i}((\langle a, d, t\rangle, |X_\ell|))$ with $(\langle a', d', t'\rangle, |X_\ell|) \subseteq \mathit{SPals}(X_\ell)$.*

Proof. By Lemma 3.4 of [13]. □

We are now ready to prove the following lemma:

Lemma 9. *For any variable $X_i = X_\ell X_r$, $\mathit{Pals}^{\triangle}(X_i)$ requires $O(\log |X_i|)$ space and can be computed in $O(n^2 \log |X_i|)$ time.*

Proof. Recall Observation 4. It is clear from the definition that $\mathit{Pals}^{*}(X_i)$ is either singleton or empty. When it is a singleton, it consists of the maximal even palindrome centered at $|X_\ell|$. Let $l = \mathit{FM}(X_r, X_\ell^R, 1)$. Then we have

$$\mathit{Pals}^{*}(X_i) = \begin{cases} \emptyset & \text{if } l = 0, \\ \{(|X_\ell| - l + 1, |X_\ell| + l)\} & \text{otherwise.} \end{cases}$$

Due to Lemma 7, $\mathit{Pals}^{*}(X_i)$ can be computed in $O(n^2)$ time.

Now we consider $\mathit{Ext}_{X_i}(\mathit{SPals}(X_\ell))$. By Lemma 7 and Lemma 8, each subset $\mathit{Ext}_{X_i}((1, \langle a, d, t\rangle)) \subseteq \mathit{Ext}_{X_i}(\mathit{SPals}(X_\ell))$ requires $O(1)$ space and can be computed in $O(n^2)$ time. It follows from Lemma 5 that $\mathit{Ext}_{X_i}(\mathit{SPals}(X_\ell))$ consists of $O(\log |X_i|)$ arithmetic progressions. Thus $\mathit{Ext}_{X_i}(\mathit{SPals}(X_\ell))$ can be computed in $O(n^2 \log |X_i|)$ time. Similar arguments hold for $\mathit{Ext}_{X_i}(\mathit{PPals}(X_r))$. □

4.3 Results

Theorem 4. *Problem 2 can be solved in $O(n^4)$ time with $O(n^2)$ space.*

Proof. Firstly we analyze the time complexity. From Theorem 3 preprocessing for the *FM* function takes $O(n^3)$ time. By Lemma 7, each call of *FM* takes $O(n^2)$ time. It follows from Lemma 9 that $Pals^{\triangle}(X_i)$ can be computed in $O(n^3)$ time. By Observation 5, $PPals(X_i)$ and $SPals(X_i)$ can be computed in $O(n)$ time from $Pals^{\triangle}(X_i)$. Hence the overall time cost to compute $\{PPals(X_i)\}_{i=1}^{n}$, $\{SPals(X_i)\}_{i=1}^{n}$, and $\{Pals^{\triangle}(X_i)_{i=1}^{n}\}$ is $O(n^4)$.

Secondly we analyze the space complexity. The preprocessing for the *FM* function requires $O(n^2)$ due to Theorem 3. From Lemma 5 $PPals(X_i)$ and $SPals(X_i)$ require $O(n)$ space. Lemma 9 states that $Pals^{\triangle}(X_i)$ requires $O(n)$ space. Thus the total space requirement is $O(n^2)$. □

The following two theorems are results obtained by slightly modifying the algorithm of the previous subsections.

Theorem 5. *Given an SLP $\mathcal{T}$ that describes string T, whether T is a palindrome or not can be determined with extra $O(1)$ space and without increasing asymptotic time complexities of the algorithm.*

Proof. It suffices to see if $(1, |T|) \in PPals(T) = PPals(X_n)$. By Lemma 5, $PPals(X_n)$ can be represented by $O(n)$ arithmetic progressions. It is not difficult to see that T is a palindrome if and only if $a + (t-1)d = |T|$ for the arithmetic progression $\langle a, d, t \rangle$ of the largest common difference among those in $PPals(X_n)$. Such an arithmetic progression can easily be found during computation of $PPals(X_n)$ without increasing asymptotic time complexities of the algorithm. □

Theorem 6. *Given an SLP $\mathcal{T}$ that describes string T, the position pair (p, q) of the longest palindrome in T can be found with extra $O(1)$ space and without increasing asymptotic time complexities of the algorithm.*

Proof. We compute the beginning and ending positions of the longest palindrome in $Pals^{\triangle}(X_i)$ for $i = 1, 2, \ldots, n$. It takes $O(n)$ time for each X_i. If its length exceeds the length of the currently kept palindrome, we update the beginning and ending positions. □

Provided that $\{PPals(X_i)\}_{i=1}^{n}$, $\{SPals(X_i)\}_{i=1}^{n}$, and $\{Pals^{\triangle}(X_i)_{i=1}^{n}\}$ are already computed, we have the following result:

Theorem 7. *Given pair (p, q) of integers, it can be answered in $O(n)$ time whether or not substring $T[p : q]$ is a maximal palindrome of T.*

Proof. We binary search the derivation tree of SLP $\mathcal{T}$ until finding the variable $X_i = X_\ell X_r$ such that $1 + \mathit{offset} \leq p \leq |X_\ell| + \mathit{offset}$ and $1 + \mathit{offset} + |X_\ell| \leq q \leq |X_i| + \mathit{offset}$. This takes $O(n)$ time. Due to Observation 4, for each variable X_i, $Pals^{\triangle}(X_i)$ can be represented by $O(n)$ arithmetic progressions plus a pair of the beginning and ending positions of a maximal palindrome. Thus, we can check if $(p, q) \in Pals^{\triangle}(X_i)$ in $O(n)$ time. □

Finally we supply pseudo-codes of our algorithms.

```
Algorithm $ComputeLCStr$
Input: SLP $\mathcal{T} = \{X_i\}_{i=1}^{n}$, $\mathcal{S} = \{Y_j\}_{j=1}^{m}$.
$L = \emptyset$;
for $i = 1 \ldots n$ do
  for $j = 1 \ldots m$ do
    compute $OL(X_i, Y_j)$ and $OL(Y_j, X_i)$;
for $i = 1 \ldots n$ do
  for $j = 1 \ldots m$ do
    if $X_i = a$ then /* $a \in \Sigma$ */
      $L = L \cup LCSuf(Y_{j_\ell}, X_i)$
            $\cup LCPref(Y_{j_r}, X_i)$;
    else if $Y_j = a$ then /* $a \in \Sigma$ */
      $L = L \cup LCSuf(X_{i_\ell}, Y_j)$
            $\cup LCPref(X_{i_r}, Y_j)$;
    else /* $X_i = X_\ell X_r$ and $Y_i = Y_\ell Y_r$ */
      $L = L \cup LCStr^*(X_i, Y_j)$
            $\cup Ext_{X_i,Y_j}(OL(X_{\ell_i}, Y_{r_j}))$
            $\cup Ext_{Y_j,X_i}(OL(Y_{\ell_j}, X_{r_i}))$;
return $\max(L)$;
```

```
Algorithm $ComputePalindromes$
Input: SLP $\mathcal{T} = \{X_i\}_{i=1}^{n}$.
for $i = 1 \ldots n$ do
    $SPals(X_i) = \emptyset$; $PPals(X_i) = \emptyset$;
    $Pals^{\triangle}(X_i) = \emptyset$;
for $i = 1 \ldots n$ do
  if $X_i = a$ then
    $SPals(X_i) = (1,1)$; $PPals(X_i) = (1,1)$;
    $Pals^{\triangle}(X_i) = (1,1)$;
  else /* $X_i = X_\ell X_r$ */
    $seed = SPals(X_\ell) \cup (PPals(X_r) \oplus |X_\ell|)$;
    $Pals^{\triangle}(X_i) = Ext_{X_i}(seed) \cup Pals^*(X_i)$;
    $SPals(X_i) = SPals(X_r)$
            $\cup \{(1,q) \in Pals^{\triangle}(X_i)\}$;
    $PPals(X_i) = PPals(X_\ell)$
            $\cup \{(p, |X_\ell|) \in Pals^{\triangle}(X_i)\}$;
return $Pals^{\triangle}(X_1), \ldots, Pals^{\triangle}(X_n)$;
```

References

1. Ziv, J., Lempel, A.: A universal algorithm for sequential data compression. IEEE Trans. Info. Theory **IT-23**(3) (1977) 337–349
2. Ziv, J., Lempel, A.: Compression of individual sequences via variable-length coding. IEEE Trans. Info. Theory **24**(5) (1978) 530–536
3. Kieffer, J., Yang, E., Nelson, G., Cosman, P.: Universal lossless compression via multilevel pattern matching. IEEE Trans. Info. Theory **46**(4) (2000) 1227–1245
4. Nevill-Manning, C.G., Witten, I.H., Maulsby, D.L.: Compression by induction of hierarchical grammars. In: DCC'94, IEEE Press (1994) 244–253
5. Gasieniec, L., Karpinski, M., Plandowski, W., Rytter, W.: Efficient algorithms for Lempel-Ziv encoding. In: SWAT'96. (1996) 392–403
6. Rytter, W.: Grammar compression, lz-encodings, and string algorithms with implicit input. In: ICALP'04. Volume 3142 of LNCS., Springer-Verlag (2004) 15–27
7. Inenaga, S., Shinohara, A., Takeda, M.: An efficient pattern matching algorithm on a subclass of context free grammars. In: DLT'04. (2004) 225–236
8. Plandowski, W.: Testing equivalence of morphisms on context-free languages. In: ESA'94. Volume 855 of Lecture Notes in Computer Science. (1994) 460–470
9. Karpinski, M., Rytter, W., Shinohara, A.: An efficient pattern-matching algorithm for strings with short descriptions. Nordic Journal of Computing **4** (1997) 172–186
10. Miyazaki, M., Shinohara, A., Takeda, M.: An improved pattern matching algorithm for strings in terms of straight-line programs. In: CPM'97. Volume 1264 of LNCS., Springer-Verlag (1997) 1–11
11. Lifshits, Y.: Processing compressed texts: A tractability border. In: CPM'07. Volume 4580 of LNCS., Springer-Verlag (2007) 228–240
12. Lifshits, Y., Lohrey, M.: Querying and embedding compressed texts. In: MFCS'06. Volume 4162 of LNCS. (2006) 681–692
13. Apostolico, A., Breslauer, D., Galil, Z.: Parallel detection of all palindromes in a string. Theoretical Computer Science **141** (1995) 163–173