Efficient Algorithms to Compute Compressed Longest Common Substrings and Compressed Palindromes

Wataru Matsubara [a] Shunsuke Inenaga [b] Akira Ishino [a 1]
Ayumi Shinohara [a] Tomoyuki Nakamura [a] Kazuo Hashimoto [a]

[a] *Graduate School of Information Sciences, Tohoku University, Japan*

[b] *Graduate School of Information Science and Electrical Engineering, Kyushu University, Japan*

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

[1] Presently at Google Japan Inc.

Email addresses: `matsubara@shino.ecei.tohoku.ac.jp` (Wataru Matsubara), `inenaga@c.csce.kyushu-u.ac.jp` (Shunsuke Inenaga), `ishino@ecei.tohoku.ac.jp` (Akira Ishino), `ayumi@ecei.tohoku.ac.jp` (Ayumi Shinohara), `nakamura@aiet.ecei.tohoku.ac.jp` (Tomoyuki Nakamura), `hk@aiet.ecei.tohoku.ac.jp` (Kazuo Hashimoto).

Preprint submitted to Elsevier Science *12 August 2009*

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 the LZ-family [13,14], run-length encoding, multi-level pattern matching code [5], Sequitur [10] and so on, can quickly be transformed into SLPs [2,12,3]. 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 [11], which tests the equality of two SLP-compressed strings in $O(n^4)$ time. Later on Karpinski et al. [4] 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. [9] and recently to $O(n^3)$ time by Lifshits [6]. 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 [4], and lately in $O(n^3 \log N)$ time [6]. Gąsieniec et al. [2] 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 [7] 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 [6] 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 was listed as an open problem in [6]. This paper closes the problem giving an algorithm that runs in $O(n^4 \log n)$ time with $O(n^3)$ space. For the second 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. [2] 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 [2], 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 [6]. 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. [2] is unclear. However, since the equality test algorithm of [6] takes $O(n^2)$ space, the above-mentioned $O(n^4 \log^2 N)$-time solution takes at least as much space as ours.

A preliminary version of this work appeared in [8].

2 Preliminaries

2.1 Notations on Strings

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 $\mathit{LCPref}(T,S)$, $\mathit{LCStr}(T,S)$, and $\mathit{LCSuf}(T,S)$ 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

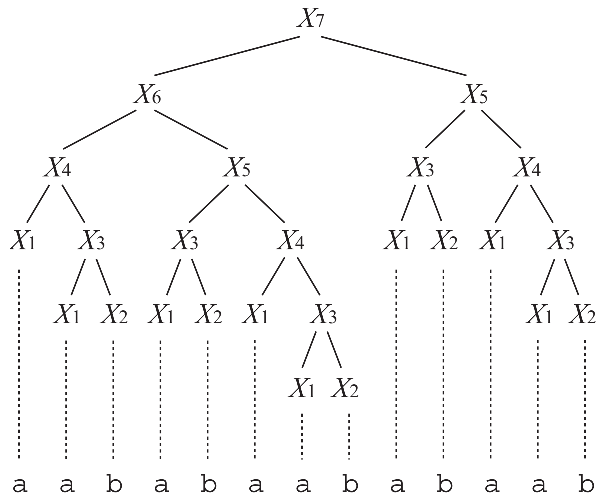

Fig. 1. The derivation tree of SLP $\mathcal{T}$ of Example 1 that generates the string $T = \texttt{aababaababaab}$.

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 $T[i : |T|]$ be a *suffix palindrome* of T.

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 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 $\mathcal{T}$. We remark that $|T| = O(2^n)$.

Example 1 *SLP* $\mathcal{T} = \{X_i\}_{i=1}^{7}$ *with* $X_1 = \texttt{a}$, $X_2 = \texttt{b}$, $X_3 = X_1X_2$, $X_4 = X_1X_3$, $X_5 = X_3X_4$, $X_6 = X_4X_5$, *and* $X_7 = X_6X_5$ *generates string* $T = \texttt{aababaababaab}$. *The derivation tree of SLP* $\mathcal{T}$ *is shown in Fig. 1.*

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 $1 \leq i \leq n$. The following lemma is important for our algorithms which will be given later on.

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 $T_n^R = T^R$. □

Example 2 *For SLP $\mathcal{T} = \{X_i\}_{i=1}^7$ of Example 1, its reversed SLP $\mathcal{T}^R = \{X_i^R\}_{i=1}^7$ consists of $X_1^R = \mathtt{a}$, $X_2^R = \mathtt{b}$, $X_3^R = X_2^R X_1^R$, $X_4^R = X_3^R X_1^R$, $X_5^R = X_4^R X_3^R$, $X_6^R = X_5^R X_4^R$, and $X_7^R = X_5^R X_6^R$. SLP $\mathcal{T}^R$ generates the reversed string $T^R = (\mathtt{aababaababaab})^R = \mathtt{baababaababaa}$.*

Note that SLP $\mathcal{T}^R$ can be easily computed from SLP $\mathcal{T}$ 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.

Example 3 *For strings X =* `ababbab` *and Y =* `babbabb`*, $OL(X,Y) = \{1,3,6\}$ since* `b`*,* `bab` *and* `babbab` *are both suffixes of X and prefixes of Y.*

Karpinski et al. [4] gave the following results for computation of OL for strings described by SLPs.

Lemma 2 ([4]) *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 ([4]) *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 $1 \leq j \leq n$.*

In order to solve Problem 1 it is useful to compute $OL(X_i, Y_j)$ and $OL(Y_j, X_i)$ for each $1 \leq i \leq n$ 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 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 position which is just one position to the left of the first position of mismathces when we compare Y_j with X_i at position k. Namely, $FM(X_i, Y_j, k)$ equals the length of the longest common prefix of $X_i[k : |X_i|]$ and Y_j;

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

Example 4 *Consider variables X_6=*`aababaab` *and X_5=*`abaab` *of Example 1. Then $FM(X_6, X_5, 2) = 3$ as $LCPref(X_6[2 : |X_6|], X_5) = LCPref($*`ababaab`*,* `abaab`*$) = 3$.*

Lemma 3 ([4]) *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 $OL(X_{i'}, Y_{j'})$ is already computed for any $1 \leq i' \leq i$ and $1 \leq j' \leq j$.*

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}.*

Example 5 *Consider SLP $\mathcal{T}$ of Example 1 generating $T =$* `aababaababaab`*. Substring* `baababaab` *of T is a substring of $X_7 = X_6 X_5$ and covers the boundary between X_6 and X_5. Substring* `baab` *of T is a substring of $X_5 = X_3 X_4$ and covers the boundary between X_3 and X_4. Substring $T[7] =$ a of T is a substring of $X_3 = X_1 X_2$ and touches the boundary between X_1 and X_2. (See also Fig. 1.)*

It directly follows from Observation 1 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 $X_i = X_{\ell_i} X_{r_i}$, $Y_j = Y_{\ell_j} Y_{r_j}$ and any non-negative integer k, let h_1 and h_2 be the maximum values such that $X_i[|X_{\ell_i}| - k - h_1 + 1 : |X_{\ell_i}| + h_2] = Y_j[|Y_{\ell_j}| - h_1 + 1 : |Y_{\ell_j}| + k + h_2]$. That is,

$$\begin{aligned} h_1 &= LCSuf(X_{\ell_i}[1 : |X_{\ell_i}| - k], Y_{\ell_j}) \text{ and} \\ h_2 &= LCPref(X_{r_i}, Y_{r_j}[k+1 : |Y_{r_j}|]). \end{aligned}$$

Then let

$$Ext_{X_i,Y_j}(k) = \begin{cases} k + h_1 + h_2 & \text{if } X_i = X_{\ell_i} X_{r_i} \text{ and } Y_j = Y_{\ell_j} Y_{r_j}, \\ k & \text{if } X_i \text{ or } Y_j \text{ is constant.} \end{cases}$$

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. 2):

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

$$\bigcup_{1 \leq i \leq n, 1 \leq j \leq m} (Ext_{X_i,Y_j}(OL(X_{\ell_i}, Y_{r_j})) \cup Ext_{Y_j,X_i}(OL(Y_{\ell_j}, X_{r_i})) \cup Ext_{X_i,Y_j}(0)),$$

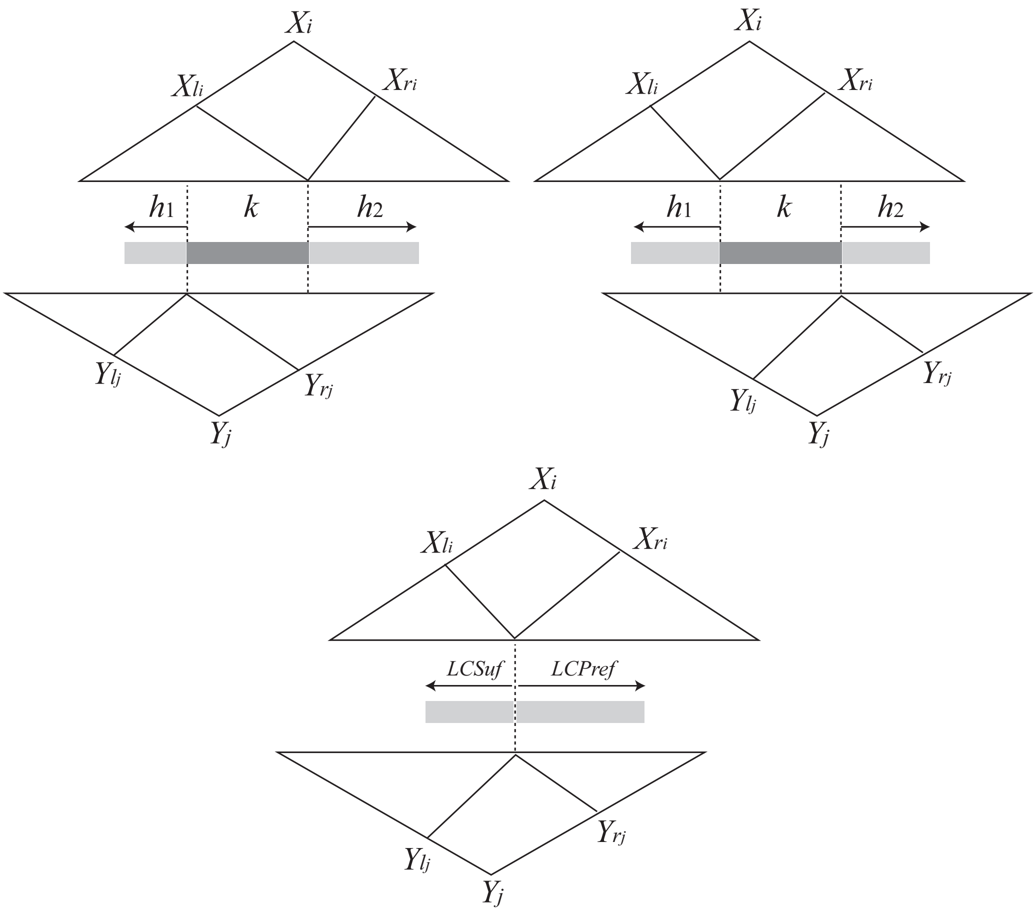

Fig. 2. Illustration of Observation 2. Each candidate for $LCStr(T,S)$ can be computed by extending either some overlap between X_{ℓ_i} and Y_{r_i} or some overlap between Y_{ℓ_i} and X_{r_i}, or concatenating $LCSuf(X_{\ell_i}, Y_{\ell_i})$ and $LCPref(X_{r_i}, Y_{r_i})$.

Based on Observation 2, our strategy for computing $LCStr(T,S)$ is to compute $\max(Ext_{X_i,Y_j}(OL(X_{\ell_i}, Y_{r_j})))$, $\max(Ext_{Y_j,X_i}(OL(Y_{\ell_j}, X_{r_i})))$, and $Ext_{X_i,Y_j}(0)$ for each pair of X_i and Y_j. Notice that $Ext_{X_i,Y_j}(0)$ can be computed in $O(n \log n)$ time due to Lemma 3, provided that the reversed SLP $\mathcal{T}^R$ and $Occ^{\triangle}(X_i^R, X_j^R)$ are already computed for each pair of variables X_i^R and X_j^R in $\mathcal{T}^R$. Lemma 4 below 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.

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

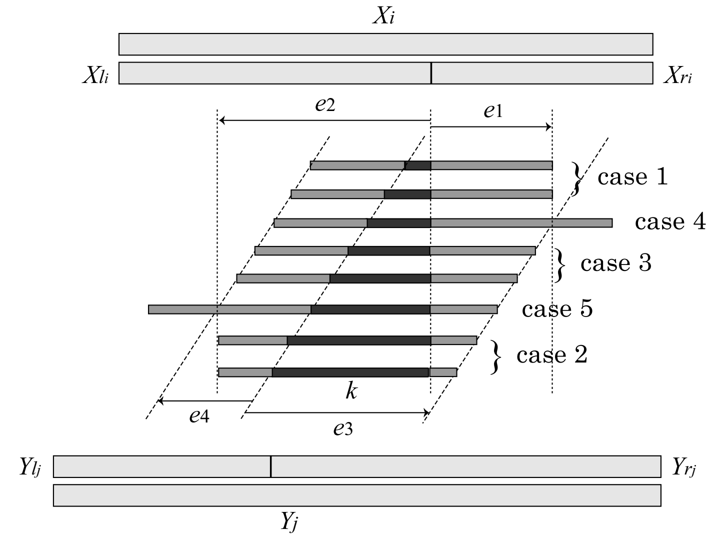

Fig. 3. Illustration for the proof of Lemma 4. The dark rectangles represent the overlaps between X_{ℓ_i} and Y_{r_j}. Case 6 is the special case where cases 4 and 5 happen at the same time and case 3 does not exist.

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, $w^* = www\cdots$.

Let e_1, e_2 be the largest integer such 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. Similarly, let e_3, e_4 be the largest integer such 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. More formally,

$$
\begin{aligned}
e_1 &= \mathit{LCPref}(X_{r_i}, (vu)^*) = \begin{cases} \mathit{FM}(Y_{r_j}, X_{r_i}, a+1) & \text{if } \mathit{FM}(Y_{r_j}, X_{r_i}, a+1) < d, \\ \mathit{FM}(X_{r_i}, X_{r_i}, d+1) + d & \text{otherwise,} \end{cases} \\
e_2 &= \mathit{LCSuf}(X_{\ell_i}, (vu)^*) = \mathit{FM}(X^R_{\ell_i}, X^R_{\ell_i}, d+1) + d, \\
e_3 &= \mathit{LCPref}(Y_{r_j}, (uv)^*) = \mathit{FM}(Y_{r_j}, Y_{r_j}, d+1) + d, \\
e_4 &= \mathit{LCSuf}(Y_{\ell_j}, (uv)^*) = \begin{cases} \mathit{FM}(X^R_{\ell_i}, Y^R_{\ell_j}, a+1) & \text{if } \mathit{FM}(X^R_{\ell_i}, Y^R_{\ell_j}, a+1) < d, \\ \mathit{FM}(Y^R_{\ell_j}, Y^R_{\ell_j}, d+1) + d & \text{otherwise.} \end{cases}
\end{aligned}
$$

(See also Fig. 3.) As above, we can compute e_1, e_2, e_3, e_4 by at most 6 calls of FM.

Let $k \in \langle a, d, t \rangle$. We categorize $\mathit{Ext}_{X_i, Y_j}(k)$ 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 $Ext_{X_i,Y_j}(k) = Ext_{X_i,Y_j}(k-d)+d$. Therefore, we have

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

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 $Ext_{X_i,Y_j}(k) = Ext_{X_i,Y_j}(k+d)+d$. Therefore, we have

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

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

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

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

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

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

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

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

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

Then clearly the following inequality stands (see also Fig. 3):

$$F \geq \max\{D, E\} \geq C \geq \max\{A, B\}. \qquad (1)$$

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 $\max(Ext_{X_i,Y_j}(\langle a,d,t\rangle))$ by at most 2 calls of FM, provided that e_1, e_2, e_3, e_4 are already computed.

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

A pseudo-code of our algorithm is given in Algorithm 1.

Algorithm 1: Computing $LCStr(T, S)$.

Input: SLPs $\mathcal{T} = \{\mathsf{X_i}\}_{\mathsf{i}=1}^{\mathsf{n}}$, $\mathcal{S} = \{\mathsf{Y_j}\}_{\mathsf{j}=1}^{\mathsf{m}}$
Output: Length of longest common substring of strings T and S

1 **for** $\mathsf{i} = 1$ **to** n **do**
2   **for** $\mathsf{j} = 1$ **to** m **do**
3   compute $\mathsf{OL}(\mathsf{X_i}, \mathsf{Y_j})$ and $\mathsf{OL}(\mathsf{Y_j}, \mathsf{X_i})$;
4
5 $\mathsf{L} = \emptyset$;
6 **for** $\mathsf{i} = 1$ **to** n **do**
7   **for** $\mathsf{j} = 1$ **to** m **do**
8         $\mathsf{L} = \mathsf{L} \cup \max(\mathsf{Ext}_{\mathsf{X_i},\mathsf{Y_j}}(\mathsf{OL}(\mathsf{X}_{\mathsf{l_i}}, \mathsf{Y}_{\mathsf{r_j}}))) \cup \max(\mathsf{Ext}_{\mathsf{Y_j},\mathsf{X_i}}(\mathsf{OL}(\mathsf{Y}_{\mathsf{l_j}}, \mathsf{X}_{\mathsf{r_i}}))) \cup \mathsf{Ext}_{\mathsf{X_i},\mathsf{Y_j}}(0)$;
9
10 **return** $\max(\mathsf{L})$;

Now we obtain the main result of this section.

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

Proof. The correctness of the algorithm is clear from lines 6-10 which correspond to Observation 2.

It follows from Theorem 1 that it takes $O(n^4 \log n)$ time and $O(n^3)$ space in lines 1-4.

For any variables $X_i = X_{\ell_i} X_{r_i}$ and $Y_j = Y_{\ell_j} Y_{r_j}$, $\max(Ext_{X_i,Y_j}(OL(X_{\ell_i}, Y_{r_j})))$ and $\max(Ext_{Y_j,X_i}(OL(Y_{\ell_j}, X_{r_i})))$ can be computed in $O(n^2 \log n)$ time by Lemma 4. Since each of $\max(Ext_{X_i,Y_j}(OL(X_{\ell_i}, Y_{r_j})))$ and $\max(Ext_{Y_j,X_i}(OL(Y_{\ell_j}, X_{r_i})))$ is singleton, we have $|L| = O(n^2)$. Hence it takes $O(n^4 \log n)$ time in lines 6-10.

Overall, the algorithm works in $O(n^4 \log n)$ time with $O(n^3)$ space. □

The following corollary is immediate from Theorem 2.

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 $\mathcal{T}$.

4.1 The Problem

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

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

Note that $|\mathit{Pals}(T)| = O(|T|) = O(2^n)$. Thus we consider a succinct representation of $\mathit{Pals}(T)$ in the sequel.

Let $\mathit{PPals}(T)$ and $\mathit{SPals}(T)$ denote the set of pairs of the beginning and ending positions of the prefix and suffix palindromes of T, respectively, that is,

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

Example 6 *For string* $T =$ `aababaababaab`*,* $\mathit{PPals}(T) = \{(1, q) \mid q \in \{1, 2, 7, 12\}\}$*, since* `a`*,* `aa`*,* `aababaa`*, and* `aababaababaa` *are prefix palindromes. Also,* $\mathit{SPals}(T) = \{(p, 13) \mid p \in \{5, 10, 13\}\}$*, since* `baababaab`*,* `baab` *and* `b` *are suffix palindromes.*

It is easy to see that for any non-empty string T, $\mathit{PPals}(T)$, $\mathit{SPals}(T)$ and $\mathit{Pals}(T)$ are non-empty sets.

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 $\mathit{Pals}^{\triangle}(X_i)$ 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,

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

Example 7 *Consider variable* $X_6 = X_4 X_5 =$ `aababaab` *of Example 1, where* $X_4 =$ `aab` *and* $X_5 =$ `abaab`*.* $\mathit{Pals}^{\triangle}(X_6) = \{(2, 4), (1, 7), (4, 6)\}$ *since* $X_6[2 : 4] =$ `aba`*,* $X_6[1 : 7] =$ `aababaa`*, and* $X_6[4 : 6] =$ `aba` *are the maximal palindromes that touch or cover the boundary of* X_4 *and* X_5*.*

We have the following observation for decomposition of $\mathit{Pals}(X_i)$ (see Fig. 4).

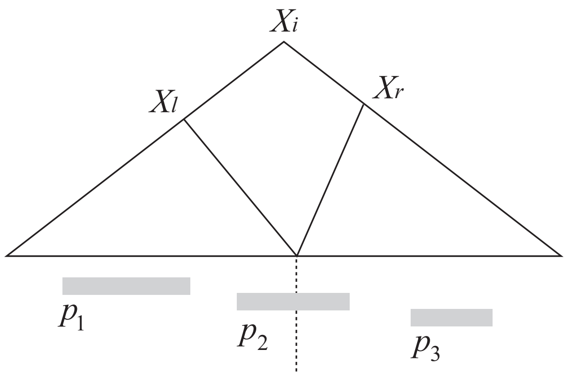

Fig. 4. Illustration of Observation 3. Any maximal palindrome of X_i is a non-suffix maximal palindrome of X_ℓ (like p_1), a maximal palindrome of X_i covering or touching the boundary of X_i (like p_2), or a non-prefix maximal palindrome of X_r (like p_3).

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

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

Thus, the desired output $Pals(T) = Pals(X_n)$ can be represented as a combination of $\{Pals^{\triangle}(X_i)\}_{i=1}^n$, $\{PPals(X_i)\}_{i=1}^n$ and $\{SPals(X_i)\}_{i=1}^n$. Therefore, computing $Pals(T)$ is reduced to computing $Pals^{\triangle}(X_i)$, $PPals(X_i)$ and $SPals(X_i)$, 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 succinct representations $\{Pals^{\triangle}(X_i)\}_{i=1}^n$, $\{PPals(X_i)\}_{i=1}^n$ and $\{SPals(X_i)\}_{i=1}^n$.*

Note that the sizes of $\{Pals^{\triangle}(X_i)\}_{i=1}^n$, $\{PPals(X_i)\}_{i=1}^n$ and $\{SPals(X_i)\}_{i=1}^n$ can be $O(2^n)$. Thus we output succinct representations of these sets which are polynomial in n. In the following sections we show how to succinctly represent and compute these sets.

4.2 Succinct Representations of $PPals(X)$ and $SPals(X)$

Gąsieniec et al. [2] claimed that $PPals(X)$ and $SPals(X)$ can be represented by $O(\log |X|)$ arithmetic progressions for any string X. However, they gave no proof regarding it. Although they stated that a proof is to be given in a full version of the paper, unfortunately it has never appeared. This section is to supply a full proof to show that $PPals(X)$ and $SPals(X)$ can be represented by $O(\log |X|)$ arithmetic progressions.

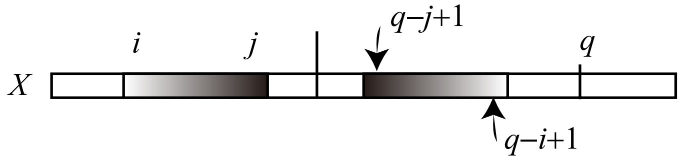

Fig. 5. $(1, q) \in PPals(X)$ implies $X[i : j] = X[q-j+1 : q-i+1]^R$.

Let us focus on the space requirement of $PPals(X)$, as that of $SPals(X)$ can be shown similarly. Recall that $PPals(X)$ is the set of pairs of the beginning and ending positions of all prefix palindromes of X.

The following lemma is obvious but is quite helpful to prove Lemma 6.

Lemma 5 *For any integers q, such that $(1, q) \in PPals(X)$ and i, j with $1 \le i < j \le q$, we have $X[i : j] = X[q - j + 1 : q - i + 1]^R$.*

Proof. Since $(1, q)$ is the prefix palindrome in X, we have $X[i] = X[q - i + 1]$ for any i with $1 \le i \le q$, which implies that:

$$\begin{aligned} X[i : j] &= X[i]\ X[i+1] \cdots X[j-1]\ X[j] \\ &= X[q-i+1]\ X[q-i] \cdots X[q-j+2]\ X[q-j+1] \\ &= (X[q-j+1]\ X[q-j+2] \cdots X[q-i]\ X[q-i+1])^R \\ &= X[q - j + 1 : q - i + 1]^R. \end{aligned}$$

(see also Fig. 5) □

Lemma 6 *For any positive integers a and d, if $(1, a), (1, a + d) \in PPals(X)$ and $a - d \ge 0$, then $(1, a - d) \in PPals(X)$.*

Proof. We show $X[1 : a - d] = X[1 : a - d]^R$, which yields that $a - d$ is the length of a prefix palindrome in X. By applying Lemma 5, we have

$$\begin{aligned} X[1 : a - d] &= X[a - (a - d) + 1 : a - 1 + 1]^R && (2) \\ &= X[d + 1 : a]^R \\ &= (X[(a + d) - a + 1 : (a + d) - (d + 1) + 1]^R)^R && (3) \\ &= X[d + 1 : a] \\ &= X[1 : a - d]^R \end{aligned}$$

where Equation (2) comes from $(1, a) \in PPals(X)$, whereas Equation (3) comes from $(1, a + d) \in PPals(X)$. (see also Fig. 6). □

Let $a_1, a_2, \ldots, a_k$ be the sequence of integers in increasing order, such that $PPals(X) = \{(1, a_1), (1, a_2), \ldots, (1, a_k)\}$. We define d_i as the progression differences for a_i, that is, $d_i = a_{i+1} - a_i$ for $1 \le i < k$. The next lemma states

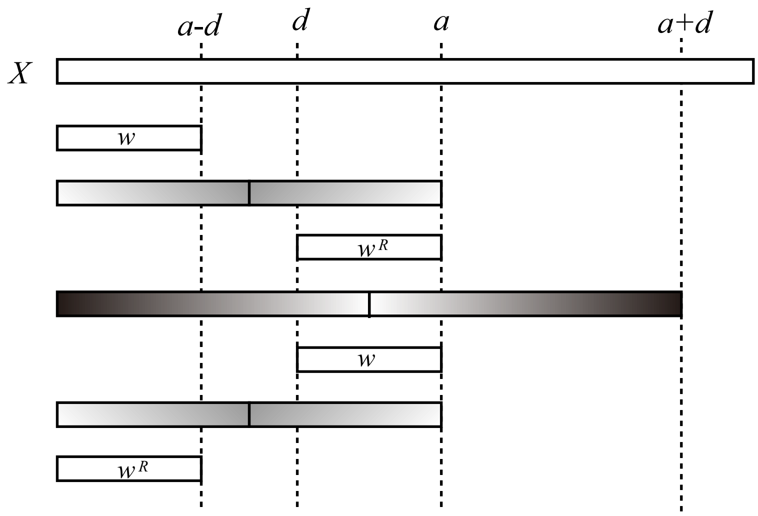

Fig. 6. $(1, a) \in PPals(X)$ and $(1, a+d) \in PPals(X)$ implies $(1, a-d) \in PPals(X)$.

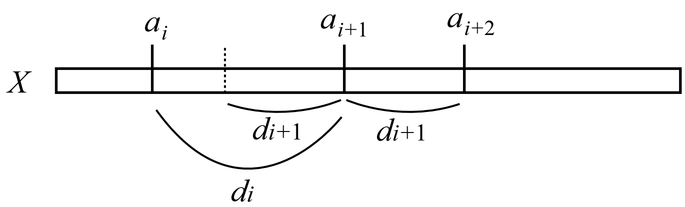

Fig. 7. $d_i > d_{i+1}$ contradicts the definition of $\{a_i\}_{i=1}^{k}$.

that the sequence $\{d_i\}_{i=1}^{k-1}$ is monotonically non-decreasing.

Lemma 7 *$d_i \leq d_{i+1}$ for any $1 \leq i < k-1$.*

Proof. Suppose $d_i > d_{i+1}$ holds for some $1 \leq i < k-1$. Since $(1, a_{i+1}) \in PPals(X)$ and $(1, a_{i+2}) = (1, a_{i+1} + d_{i+1}) \in PPals(X)$, Lemma 6 claims that $(1, a_{i+1} - d_{i+1}) \in PPals(X)$. However, $a_i = a_{i+1} - d_i < a_{i+1} - d_{i+1} < a_{i+1}$, which contradicts the definition that $(1, a_{i+1})$ is the next element to $(1, a_i)$ in $PPals(X)$ in increasing order (see also Fig. 7). $\square$

Lemma 8 *If $d_{i+1} \neq d_i$, then $d_{i+1} \geq d_i + d_{i-1}$.*

Proof. By Lemma 6, we have $(1, a_{i+1} - d_i) \in PPals(X)$ since $(1, a_{i+1}) \in PPals(X)$ and $(1, a_{i+2}) = (1, a_{i+1} + d_{i+1}) \in PPals(X)$. Therefore, $a_{i+1} - d_{i+1} = a_j$ for some $1 \leq j \leq i$, so that $d_{i+1} = a_{i+1} - a_j = \sum_{\ell=j}^{i}(a_{\ell+1} - a_\ell) = \sum_{\ell=j}^{i} d_\ell$. If $d_{i+1} \neq d_i$, we have $j < i$, which implies $d_{i+1} = \sum_{\ell=j}^{i} d_\ell \geq d_i + d_{i-1}$. $\square$

The following is a key lemma of this subsection:

Lemma 9 *For any variable X, $PPals(X)$ and $SPals(X)$ can be represented by $O(\log |X|)$ arithmetic progressions.*

Proof. We show that $PPals(X)$ can be represented by $O(\log|X|)$ arithmetic progressions. The case of $SPals(X)$ can be proved similarly.

It follows from Lemma 6 that, for any positive integer r such that $a_i - rd_i > 0$, we have $a_i - rd_i \in PPals(X)$. For any a_i and d_i, let $t_i = \max\{y \mid a_i - (y-1)d_i > 0\}$ and $a'_i = a_i - (t_i - 1)d_i$. That is, a'_i is the smallest element of arithmetic progression $\langle a'_i, d_i, t_i \rangle$. Then, if $d_i = d_{i+1}$, it holds that $\langle a'_i, d_i, t_i \rangle \cup \{a_{i+1}\} = \langle a'_{i+1}, d_{i+1}, t_{i+1} \rangle$. For any integers p, q and any arithmetic progression $\langle a, d, t \rangle$ such that $p \leq a$ and $q \geq a + (t-1)d$, let

$$(p, \langle a, d, t \rangle) = \{(p, a + (i-1)d) \mid 1 \leq i \leq t\}, \text{ and}$$
$$(\langle a, d, t \rangle, q) = \{(a + (i-1)d, q) \mid 1 \leq i \leq t\}.$$

Then we have $PPals(X) = \bigcup_{1 \leq i \leq n}(1, \langle a'_i, d_i, t_i \rangle) = \bigcup_{i \in \{i \mid d_i \neq d_{i+1}\}}(1, \langle a'_i, d_i, t_i \rangle)$. The worst case scenario in terms of the number of arithmetic progressions in $PPals(X)$ is that $d_i \neq d_{i+1}$ for each i. By Lemma 8, the actual worst case is given by the following sequence $\{d_i\}_{i=1}^{k-1}$:

$$d_i = \begin{cases} 2 & \text{for } i = 1, \\ 3 & \text{for } i = 2, \\ d_{i-1} + d_{i-2} & \text{for } i > 2. \end{cases}$$

Now, let F_j denote the j-th Fibonacci number, namely,

$$F_j = \begin{cases} 1 & \text{for } j = 1, 2, \\ F_{j-1} + F_{j-2} & \text{for } j > 2. \end{cases}$$

It is a well-known fact that $F_i = \frac{\varphi^i - (1-\varphi)^i}{\sqrt{5}} = \lfloor \frac{\varphi^i}{\sqrt{5}} + \frac{1}{2} \rfloor$, where $\varphi = \frac{\sqrt{5}+1}{2}$.

Clearly $d_i = F_{i+2}$. Therefore, the general term of $\{a_i\}$ can be represented as follows:

$$\begin{aligned} a_i &= a_{i-1} + d_{i-1} = a_{i-2} + d_{i-2} + d_{i-1} \cdots = a_1 + \sum_{k=1}^{i-1} d_k = a_1 + \sum_{k=3}^{i+1} F_k \\ &= a_1 + \sum_{k=1}^{i+1} F_k - F_1 - F_2 = 1 + F_{i+1+2} - 1 - 1 - 1 = F_{i+3} - 2. \end{aligned}$$

Now we have the following formula for the largest element a_k of $\{a_i\}_{i=1}^{k}$.

$$a_k = F_{k+3} - 2 = \lfloor \frac{\varphi^{k+3}}{\sqrt{5}} + \frac{1}{2} \rfloor - 2 > \frac{\varphi^{k+3}}{\sqrt{5}} + \frac{1}{2} - 1 - 2.$$

Since $a_k \leq |X|$ and $\varphi > 1$, we have that $k = O(\log_\varphi |X|) = O(\log |X|)$. □

4.3 Efficient Computation of $Pals^{\triangle}(X_i)$, $PPals(X_i)$ and $SPals(X_i)$

In this section we show how to efficiently compute $Pals^{\triangle}(X_i)$, $PPals(X_i)$ and $SPals(X_i)$.

The next lemma points out that $SPals(X_\ell)$ and $PPals(X_r)$ are useful to compute $Pals^{\triangle}(X_i)$.

Lemma 10 *For any variable $X_i = X_\ell X_r$ and any $(p, q) \in Pals^{\triangle}(X_i)$, there exists an integer $l \geq 0$ such that $(p + l, q - l) \in SPals(X_\ell) \cup (PPals(X_r) \oplus |X_\ell|) \cup \{(|X_\ell|, |X_\ell| + 1)\}$.*

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 $(p + l, q - l) \in SPals(X_\ell)$.
(2) When $\lfloor \frac{p+q}{2} \rfloor > |X_\ell|$, for $l = |X_\ell| - p + 1$, we have $(p + l, q - l) \in PPals(X_r)$.
(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 $(|X_\ell|, |X_\ell|) \in SPals(X_\ell)$. 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 10, $Pals^{\triangle}(X_i)$ can be computed by "extending" all palindromes in $SPals(X_\ell)$ and $PPals(X_r)$ 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} as

$$Ext_{X_i}(p, q) = (p - h, q + h),$$

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 $Ext_{X_i}(p, q)$ undefined. There are the following natural properties on function Ext_{X_i}:

- the input and output palindromes are centered at the same position;
- if $|P| = q - p + 1$ is odd, then $Ext_{X_i}(p, q)$ is also an odd palindrome;
- if $|P| = q - p + 1$ is even, then $Ext_{X_i}(p, q)$ is also an even palindrome.

For a set S of pairs of integers, let $Ext_{X_i}(S) = \{Ext_{X_i}(p, q) \mid (p, q) \in S\}$.

Let

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

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

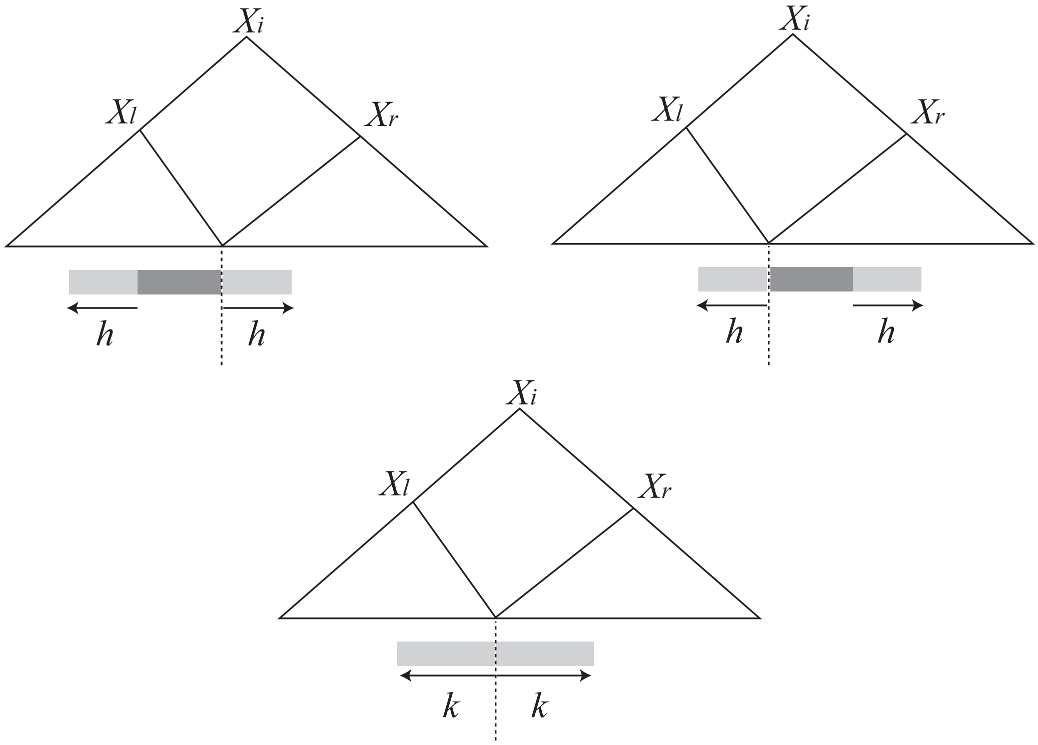

Fig. 8. Illustration of Observation 4. Any element of $Pals(X_i)$ can be computed by extending either some prefix palindrome of $SPals(X_\ell)$ or some suffix palindrome of $PPals(X_r)$, or it is the maximal even palindrome centered at $|X_\ell|$ in X_i.

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

$$Pals^{\triangle}(X_i) = Ext_{X_i}(SPals(X_\ell)) \cup Ext_{X_i}(PPals(X_r) \oplus |X_\ell|) \cup Pals^*(X_i). \quad (4)$$

See also Fig. 8 that illustrates Observation 4.

In what follows we show how to efficiently execute the *Ext* functions in Equation (4). Let us first briefly recall the work of [9,6]. For any variables $X_i = X_\ell X_r$ and X_j, we define the set $Occ^{\triangle}(X_i, X_j)$ of all occurrences of X_j that cover or touch the boundary between X_ℓ and X_r, namely,

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

Theorem 3 ([6]) *For any variables X_i and X_j, $Occ^{\triangle}(X_i, X_j)$ can be computed in total of $O(n^3)$ time and $O(n^2)$ space.*

Lemma 11 ([9]) *For any variables X_i, X_j and integer k, $FM(X_i, X_j, k)$ can be computed in $O(n^2)$ time, provided that $Occ^{\triangle}(X_{i'}, X_{j'})$ is already computed for any $1 \leq i' \leq i$ and $1 \leq j' \leq j$.*

Lemma 12 *For any variable $X_i = X_\ell X_r$ and any arithmetic progression $\langle a, d, t\rangle$ with $(1, \langle a, d, t\rangle) \subseteq PPals(X_r)$, $Ext_{X_i}((1, \langle a, d, t\rangle))$ 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. The same holds for any arithmetic progression* $\langle a', d', t' \rangle$ *with* $(\langle a', d', t' \rangle, |X_\ell|) \subseteq SPals(X_\ell)$.

The above lemma can be inherently proven by Lemma 3.4 of [1]. However, for the sake of completeness we supply a full proof of the lemma in Appendix.

We are now ready to prove the following lemma:

Lemma 13 *For any variable* $X_i = X_\ell X_r$, $Pals^{\triangle}(X_i)$ *can be represented by* $O(\log |X_i|)$ *arithmetic progressions and can be computed in* $O(n^2 \log |X_i|)$ *time.*

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

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

Due to Lemma 11, $Pals^*(X_i)$ can be computed in $O(n^2)$ time.

Now we consider $Ext_{X_i}(PPals(X_\ell))$. It follows from Lemma 12 that each subset $Ext_{X_i}((1, \langle a, d, t \rangle)) \subseteq Ext_{X_i}(PPals(X_\ell))$ can be represented by $O(1)$ arithmetic progressions. Also, $Ext_{X_i}((1, \langle a, d, t \rangle))$ can be computed in $O(n^2)$ time due to Lemma 11 and Lemma 12. It follows from Lemma 9 that $PPals(X_\ell)$ consists of $O(\log |X_\ell|)$ arithmetic progressions. Thus $Ext_{X_i}(PPals(X_\ell))$ can be computed in $O(n^2 \log |X_\ell|)$ time. Similar arguments hold for $Ext_{X_i}(SPals(X_r))$.

Hence, by Observation 4, $Pals^{\triangle}(X_i)$ can be represented by $O(\log |X_i|)$ arithmetic progressions and can be computed in $O(n^2 \log |X_i|)$ time. □

On the other hand, $PPals(X_i)$ and $SPals(X_i)$ can be computed using $Pals^{\triangle}(X_i)$, as follows:

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

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

See also Fig. 9 that illustrates Observation 5.

Lemma 14 *For any SLP variable* $X_i = X_\ell X_r$, $PPals(X_i)$ *and* $SPals(X_i)$ *can be computed in* $O(\log |X_i|)$ *time, provided that* $PPals(X_\ell)$, $SPals(X_r)$ *and* $Pals^{\triangle}(X_i)$ *are already computed.*

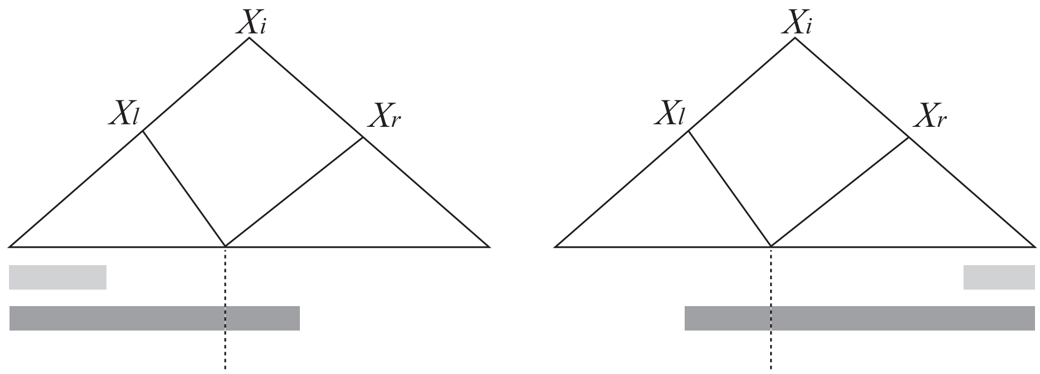

Fig. 9. Illustration of Observation 5. Any element of $PPals(X_i)$ is either an element of $PPals(X_\ell)$ or an element of $Pals^\triangle(X_i)$ whose beginning position is 1. Similar arguments hold for $SPals(X_i)$.

Proof. Clear from Lemma 9 and Lemma 13. □

4.4 Results

Algorithm 2 shows a pseudo-code of our algorithm that computes a succinct representation of all maximal palindromes of a given SLP-compressed string.

Algorithm 2: Computing succinct representation of $Pals(T)$.

Input: SLP $\mathcal{T} = \{\mathsf{X_i}\}_{\mathsf{i}=1}^{\mathsf{n}}$

Output: Succinct representation of $Pals(\mathsf{T})$ for string T

for $\mathsf{i} = 1$ **to** n **do**
    for $\mathsf{j} = 1$ **to** n **do**
        compute $Occ^\triangle(\mathsf{X_i}, \mathsf{X_j})$;

for $\mathsf{i} = 1$ **to** n **do**
    $SPals(\mathsf{X_i}) = \emptyset$; $PPals(\mathsf{X_i}) = \emptyset$; $Pals^\triangle(\mathsf{X_i}) = \emptyset$;
for $\mathsf{i} = 1$ **to** n **do**
    if $\mathsf{X_i} = \mathsf{a}$ **then** /* $\mathsf{X_i}$ is constant */
        $SPals(\mathsf{X_i}) = \langle 1, 1, 1\rangle$; $PPals(\mathsf{X_i}) = \langle 1, 1, 1\rangle$; $Pals^\triangle(\mathsf{X_i}) = \langle 1, 1, 1\rangle$;
    else /* $\mathsf{X_i} = \mathsf{X_l X_r}$ */
        $Pals^\triangle(\mathsf{X_i}) = Ext_{\mathsf{X_i}}(SPals(\mathsf{X_l})) \cup Ext_{\mathsf{X_i}}(PPals(\mathsf{X_r}) \oplus |\mathsf{X_l}|) \cup Pals^*(\mathsf{X_i})$;
        $PPals(\mathsf{X_i}) = PPals(\mathsf{X_l}) \cup \{(\mathsf{p}, |\mathsf{X_l}|) \in Pals^\triangle(\mathsf{X_i})\}$;
        $SPals(\mathsf{X_i}) = (SPals(\mathsf{X_r}) \oplus |\mathsf{X_l}|) \cup \{(1, \mathsf{q}) \in Pals^\triangle(\mathsf{X_i})\}$;

return $\{Pals^\triangle(\mathsf{X_i})\}_{\mathsf{i}=1}^{\mathsf{n}}$, $\{SPals(\mathsf{X_i})\}_{\mathsf{i}=1}^{\mathsf{n}}$, $\{PPals(\mathsf{X_i})\}_{\mathsf{i}=1}^{\mathsf{n}}$;

The main result of this section is the following theorem.

Theorem 4 *Algorithm 2 solves Problem 2 in $O(n^4)$ time with $O(n^2)$ space.*

Proof. The correctness of the algorithm follows from lines 11-13 that correspond to Observations 4 and 5.

Now we analyze the time complexity. It follows from Theorem 3 that it takes $O(n^3)$ time in total for lines 1-4. By Lemma 13 it takes $O(n^2 \log |X_i|)$ time for line 11. Also, by Lemma 14 it takes $O(\log |X_i|)$ time for lines 12-13. Therefore the time complexity for the **for** loop of line 7 is $O(n^4)$. Hence the overall time cost is $O(n^4)$.

The total space complexity is as follows. It follows from Theorem 3 that it takes $O(n^2)$ space for lines 1-4. By Lemma 13, it takes $O(\log |X_i|)$ space for line 11. Also, by Lemma 9, it takes $O(\log |X_i|)$ space for lines 12-13. Therefore the space complexity for the **for** loop of line 7 is $O(n^2)$. Hence the overall 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 $(1, |T|) \in \mathit{PPals}(T) = \mathit{PPals}(X_n)$. By Lemma 9, $\mathit{PPals}(X_n)$ 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 $\mathit{PPals}(X_n)$. Such an arithmetic progression can easily be found during computation of $\mathit{PPals}(X_n)$ 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 $\mathit{Pals}^{\triangle}(X_i)$ 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 $\{\mathit{PPals}(X_i)\}_{i=1}^{n}$, $\{\mathit{SPals}(X_i)\}_{i=1}^{n}$, and $\{\mathit{Pals}^{\triangle}(X_i)_{i=1}^{n}\}$ are already computed, we have the following result:

Theorem 7 *Given a 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 $1 + \mathit{offset} \le p \le |X_\ell| + \mathit{offset}$ and $1 + \mathit{offset} + |X_\ell| \le q \le |X_i| + \mathit{offset}$. This takes $O(n)$ time. Due to Observation 4, for each variable X_i, $\mathit{Pals}^\triangle(X_i)$ 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 $(p, q) \in \mathit{Pals}^\triangle(X_i)$ in $O(n)$ time. □

5 Conclusions and Further Work

In this paper we considered strings compressed by straight line programs (SLPs). Since SLP-compressed strings can be exponentially small w.r.t. the uncompressed (original) strings, it is significant to process SLP-compressed strings without decompression and in time polynomial in the compressed size n. In this paper, we showed the first polynomial time algorithm to compute the longest common substring of two given SLP-compressed strings, which runs in $O(n^4 \log n)$ time and $O(n^3)$ space. In addition, we presented an $O(n^4)$-time $O(n^2)$-space algorithm to compute all maximal palindromes of a given SLP-compressed strings. This is faster than the $O(n^4 \log N)$-time solution obtained by combining the results of Gąsieniec et al. [2] and Lifshits [6].

Our future work includes extending our results to computing all *squares* from a given SLP-compressed string. Gąsieniec et al. [2] claimed that all squares can be found in $O(n^6 \log^5 N)$ time from strings compressed by compositions systems, which are generalization of SLPs. The time complexity would be improved to $O(n^5 \log^3 N)$ in combination with the algorithm by Lifshits [6]. Still, it might be possible to produce a faster solution using the techniques presented in this paper.

References

[1] A. Apostolico, D. Breslauer, and Z. Galil. Parallel detection of all palindromes in a string. *Theoretical Computer Science*, 141:163–173, 1995.

[2] L. Gasieniec, M. Karpinski, W. Plandowski, and W. Rytter. Efficient algorithms for Lempel-Ziv encoding. In *Proc. 5th Scandinavian Workshop on Algorithm Theory (SWAT'96)*, volume 1097 of *Lecture Notes in Computer Science*, pages 392–403. Springer-Verlag, 1996.

[3] S. Inenaga, A. Shinohara, and M. Takeda. An efficient pattern matching algorithm on a subclass of context free grammars. In *Proc. Eighth International Conference on Developments in Language Theory (DLT'04)*, volume 3340 of *Lecture Notes in Computer Science*, pages 225–236. Springer-Verlag, 2004.

[4] M. Karpinski, W. Rytter, and A. Shinohara. An efficient pattern-matching algorithm for strings with short descriptions. *Nordic Journal of Computing*, 4:172–186, 1997.

[5] J. Kieffer, E. Yang, G. Nelson, and P. Cosman. Universal lossless compression via multilevel pattern matching. *IEEE Transactions on Information Theory*, 46(4):1227–1245, 2000.

[6] Y. Lifshits. Processing compressed texts: A tractability border. In *Proc. 18th Annual Symposium on Combinatorial Pattern Matching (CPM'07)*, volume 4580 of *Lecture Notes in Computer Science*, pages 228–240. Springer-Verlag, 2007.

[7] Y. Lifshits and M. Lohrey. Querying and embedding compressed texts. In *Proc. 31st International Symposium on Mathematical Foundations of Computer Science (MFCS'06)*, volume 4162 of *Lecture Notes in Computer Science*, pages 681–692. Springer-Verlag, 2006.

[8] W. Matsubara, S. Inenaga, A. Ishino, A. Shinohara, T. Nakamura, and K. Hashimoto. Computing longest common substring and all palindromes from compressed strings. In *Proc. 34th International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM'08)*, volume 4910 of *Lecture Notes in Computer Science*, pages 364–375. Springer-Verlag, 2008.

[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)*, volume 1264 of *Lecture Notes in Computer Science*, pages 1–11. Springer-Verlag, 1997.

[10] C. G. Nevill-Manning, I. H. Witten, and D. L. Maulsby. Compression by induction of hierarchical grammars. In *Data Compression Conference '94*, pages 244–253. IEEE Computer Society, 1994.

[11] W. Plandowski. Testing equivalence of morphisms on context-free languages. In *Proc. Second Annual European Symposium on Algorithms (ESA'94)*, volume 855 of *Lecture Notes in Computer Science*, pages 460–470. Springer-Verlag, 1994.

[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)*, volume 3142 of *Lecture Notes in Computer Science*, pages 15–27. Springer-Verlag, 2004.

[13] J. Ziv and A. Lempel. A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory*, IT-23(3):337–349, 1977.

[14] J. Ziv and A. Lempel. Compression of individual sequences via variable-length coding. *IEEE Transactions on Information Theory*, 24(5):530–536, 1978.

Appendix

This appendix is to give a complete proof for Lemma 12. To prove this lemma, we need to show the following lemma:

Lemma 15 *For any variable X_i and $\{(1, q) \mid q \in \langle a, d, t\rangle\} \subseteq PPals(X_i)$, there exist palindromes u, v and a non-negative integer k, such that $(uv)^{t+k-1}u$ is a prefix of X_i, $|uv| = d$ and $|(uv)^k u| = a$.*

Proof. Let $k = \max\{h \mid a - hd > 0\}$, $a' = a - kd$. It is not difficult to see that $\langle a', d, t + k\rangle \subseteq PPals(X_i)$. Let $w = X_i[1 : d]$, $u = X_i[1 : a']$, and $v = X_i[a' + 1 : d]$. Then, $a = a' + kd = |u| + k|uv| = |(uv)^k u|$.

Since $(1, a' + d) \in PPals(X_i)$, $X_i[d + 1 : a' + d] = u^R$. Also, for any $1 \le j \le t + k - 1$, since $(1, a' + jd) \in PPals(X_i)$, we have

$$X_i[a' + jd + 1 : a' + (j + 1)d] = w^R.$$

Thus $uvu^R(w^R)^{t+k-2}$ is a prefix of X_i.

Since $(1, a') \in PPals(X_i)$, u is a palindrome. Since $(1, a' + d) \in PPals(X_i)$, uvu^R is a palindrome, which implies that v is also a palindrome. Consequently,

$$\begin{aligned} uvu^R(w^R)^{t+k-2} &= uvu((uv)^R)^{t+k-2} = uvu(v^R u^R)^{t+k-2} \\ &= uvu(vu)^{t+k-2} = u(vu)^{t+k-1} = (uv)^{t+k-1}u. \end{aligned}$$

Therefore, $(uv)^{t+k-1}u$ is a prefix of X_i. □

In the above lemma, clearly $|uv| = d$ is a period of string $(uv)^t u$.

We are now ready to prove Lemma 12. (See also Fig. 10.)

Proof. Let us consider $Ext_{X_i}(\{1, \langle a, d, t\rangle\})$. By Lemma 15, $X_r[1 : a+(t-1)d] = (uv)^{t+k-1}u$, where $|uv| = d$ and $|(uv)^k u| = a$. Let x be the maximum integer such that $X_r[1 : x]$ has a period $|uv|$. Namely, $X_r[1 : x]$ is the longest prefix of X_r that has a period $|uv|$. Then x can be computed by using FM as follows:

$$x = FM(X_r, X_r, d + 1) + d.$$

Let y be the largest integer such that $(uv)^y$ is a prefix of X_ℓ^R. Then y can be computed by at most 2 calls of FM, as follows. First, we call FM to check whether or not the string uv is a prefix of X_ℓ^R. If $FM(X_r, X_\ell^R, 1) < d$, then $y = FM(X_r, X_\ell^R, 1)$. Otherwise, by Lemma 1 we can compute y by:

$$y = FM(X_\ell^R, X_\ell^R, d + 1) + d.$$

Let $e_\ell = |X_\ell| - y + 1$ and $e_r = |X_\ell| + x$. Then, clearly string $X_i[e_\ell : e_r]$ has a period d. Let

$$\begin{aligned}\langle a, d, t\rangle &= \langle a_1, d, t_1\rangle \cup \langle a_2, d, t_2\rangle \cup \langle a_3, d, t_3\rangle \\ &= \langle a, d, t_1\rangle \cup \langle a + t_1 d, d, t_2\rangle \cup \langle a + (t_1 + t_2)d, d, t_3\rangle, \text{ such that}\end{aligned}$$

$$\begin{aligned}&|X_\ell| - e_\ell + 1 < e_r - q_1 \text{ for any } q_1 \in \langle a_1, d, t_1\rangle,\\ &|X_\ell| - e_\ell + 1 = e_r - q_2 \text{ for any } q_2 \in \langle a_2, d, t_2\rangle,\\ &|X_\ell| - e_\ell + 1 > e_r - q_3 \text{ for any } q_3 \in \langle a_3, d, t_3\rangle,\end{aligned}$$

and $t_1 + t_2 + t_3 = t$. For the first and the last arithmetic progressions, we have:

$$\begin{aligned}Ext_{X_i}((1, \langle a_1, d, t_1\rangle)) &= \{(e_\ell, q_1 + |X_\ell| - e_\ell + 1) \mid q_1 \in \langle a_1, d, t_1\rangle\}\\ &= \{(e_\ell, \langle a + |X_\ell| - e_\ell + 1, d, t_1\rangle\} \text{ and}\\ Ext_{X_i}((1, \langle a_3, d, t_3\rangle)) &= \{(|X_\ell| + e_r - q_3, |X_\ell| + e_r) \mid q_3 \in \langle a_3, d, t_3\rangle\}\\ &= \{(\langle |X_\ell| + e_r - a - (t-1)d, d, t_3\rangle, |X_\ell| + e_r)\}.\end{aligned}$$

Now let us consider $\langle a + t_1 d, d, t_2\rangle$. It is easy to see that $t_2 \leq 1$. We consider the case where $t_2 = 1$ and $a_2 = a + t_1 d = q_2$. Notice that the palindrome $(1, a_2)$ can be expanded beyond the periodicity w.r.t. d. Thus,

$$Ext_{X_i}((1, a_2)) = \{(|X_\ell| - z + 1, |X_\ell| + a_2 + z)\} = \{(|X_\ell| - z + 1, |X_\ell| + a + t_1 d + z)\},$$

where $z = FM(X_\ell^R, X_r, a_2+1) + a_2$. Therefore, the set of expanded palindromes can be represented as follows:

$$\begin{aligned}Ext_{X_i}(\{1, \langle a, d, t\rangle\} \oplus |X_\ell|) = &\{(e_\ell, \langle a + |X_\ell| - e_\ell + 1, d, t_1\rangle\}\\ &\cup\{(\langle |X_\ell| + e_r - a - (t-1)d, d, t_3\rangle, |X_\ell| + e_r)\}\\ &\cup\{(|X_\ell| - z + 1, |X_\ell| + a + t_1 d + z)\}.\end{aligned}$$

Hence $Ext_{X_i}(\{1, \langle a, d, t\rangle\})$ can be represented by at most 2 arithmetic progressions and a palindrome, which in total require a constant space. We remark that similar arguments hold for $Ext_{X_i}(\langle a', d', t'\rangle, |X_\ell|)$. $\square$

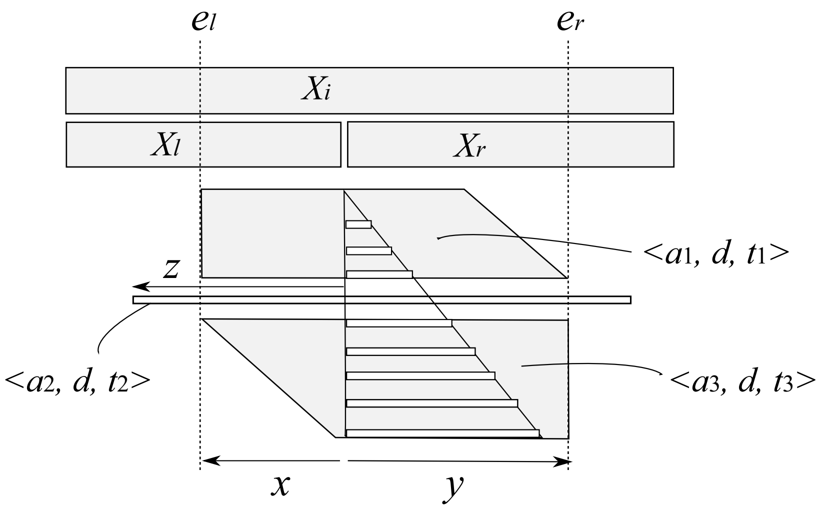

Fig. 10. Illustration for the proof of Lemma 12.