Minimal unique palindromic substrings after single-character substitution

Mitsuru Funakoshi\textsuperscript{1,2} and Takuya Mieno\textsuperscript{1,2}

\textsuperscript{1}Department of Informatics, Kyushu University. \{mitsuru.funakoshi,takuya.mieno\}@inf.kyushu-u.ac.jp
\textsuperscript{2}Japan Society for the Promotion of Science.

\section*{Abstract}

A palindrome is a string that reads the same forward and backward. A palindromic substring \(w\) of a string \(T\) is called a minimal unique palindromic substring (MUPS) of \(T\) if \(w\) occurs only once in \(T\) and any proper palindromic substring of \(w\) occurs at least twice in \(T\). MUPSs are utilized for answering the shortest unique palindromic substring problem, which is motivated by molecular biology [Inoue et al., 2018]. Given a string \(T\) of length \(n\), all MUPSs of \(T\) can be computed in \(O(n)\) time. In this paper, we study the problem of updating the set of MUPSs when a character in the input string \(T\) is substituted by another character.

We first analyze the number \(d\) of changes of MUPSs when a character is substituted, and show that \(d\) is in \(O(\log n)\). Further, we present an algorithm that uses \(O(n)\) time and space for preprocessing, and updates the set of MUPSs in \(O(\log \sigma + (\log \log n)^2 + d)\) time where \(\sigma\) is the alphabet size. We also propose a variant of the algorithm, which runs in optimal \(O(1 + d)\) time when the alphabet size is constant.

\section{Introduction}

Palindromes are strings that read the same forward and backward. Finding palindromic structures has important applications to analyze biological data such as DNA, RNA, and proteins, and thus algorithms and combinatorial properties on palindromic structures have been heavily studied (e.g., see [21, 11, 24, 17, 25] and references therein). In this paper, we treat a notion of palindromic structures called minimal unique palindromic substring (MUPS) that is introduced in [19]. A palindromic substring \(T[i..j]\) of a string \(T\) is called a MUPS of \(T\) if \(T[i..j]\) occurs exactly once in \(T\) and \(T[i+1..j-1]\) occurs at least twice in \(T\). MUPSs are utilized for solving the shortest unique palindromic substring (SUPS) problem proposed by Inoue et al. [19], which is motivated by an application in molecular biology. They showed that there are no more than \(n\) MUPSs in any length-\(n\) string, and proposed an \(O(n)\)-time algorithm to compute all MUPSs of a given string of length \(n\) over an integer alphabet of size \(n^{O(1)}\). After that, Watanabe et al. [28] considered the problem of computing MUPSs in an run-length encoded (RLE) string. They showed that there are no more than \(m\) MUPSs in a string whose RLE size is \(m\). Also, they proposed an \(O(m \log \sigma_R)\)-time and \(O(m)\)-space algorithm to compute all MUPSs of a string given in RLE, where \(\sigma_R\) is the number of distinct single-character runs in the RLE string. Recently, Mieno et al. [23] considered the problems of computing palindromic structures in the sliding window model. They showed that the set of MUPSs in a sliding window can be maintained in a total of \(O(n \log \sigma_W)\) time and \(O(D)\) space while a window of size \(D\) shifts over a string of length \(n\) from the left-end to the right-end, where \(\sigma_W\) is the maximum number of distinct characters in the windows. This result can be rephrased as follows: The set of MUPSs in a string of length \(D\) can be updated in amortized \(O(\log \sigma_W)\) time using \(O(D)\) space after deleting the first character or inserting a character to the right-end.

\textsuperscript{*}This is a full version of a paper presented at SPIRE 2021 [14].
To the best of our knowledge, there is no efficient algorithm for updating the set of MUPSs after editing a character at any position so far. Now, we consider the problem of updating the set of MUPSs in a string after substituting a character at any position. Formally, we tackle the following problem: Given a string $T$ of length $n$ over an integer alphabet of size $n^{O(1)}$ to preprocess, and then given a query of single-character substitution. Afterwards, we return the set of MUPSs of the edited string. In this paper, we first show that the number of changes of MUPSs after a single-character substitution is $O(\log n)$. In addition, we present an algorithm that uses $O(n)$ time and space for preprocessing, and updates the set of MUPSs in $O(\log \sigma + (\log \log n)^2 + d) \subset O(\log n)$ time. We also propose a variant of the algorithm, which runs in optimal $O(1 + d)$ time when the alphabet size is constant.

Related Work.

There are some results for the problem of computing string regularities, including palindromes, on dynamic strings [3, 2, 3, 9]. In this work, we consider the problem of computing MUPSs after a single edit operation as a first step toward a fully dynamic setting. This line of research was initiated by Amir et al. [14], who tackled the problem of computing the longest common factor after one edit. After that, other notions of string regularities are treated in a similar setting [27, 1, 15]. In particular, regarding palindromic structures, Funakoshi et al. [15] proposed algorithms for computing the longest palindromic substring after single-character or block-wise edit operations.

Paper Organization.

This paper is organized as follows. Section 2 introduces necessary notations and algorithmic tools. Section 3 analyzes the changes of the set of MUPSs after a single-character substitution. We then present an algorithm for updating the set of MUPSs after a single-character substitution in Section 4. Finally, we conclude in Section 5.

2 Preliminaries

2.1 Notations

Strings.

Let $\Sigma$ be an alphabet of size $\sigma$. An element of $\Sigma$ is called a character. An element of $\Sigma^*$ is called a string. The length of a string $T$ is denoted by $|T|$. The empty string $\varepsilon$ is the string of length 0. For a string $T = xyz$, then $x, y$, and $z$ are called a prefix, substring, and suffix of $T$, respectively. They are called a proper prefix, proper substring, proper suffix of $T$ if $x \neq T$, $y \neq T$, and $z \neq T$, respectively. For each $1 \leq i \leq |T|$, $T[i]$ denotes the $i$-th character of $T$. For each $1 \leq i \leq j \leq |T|$, $T[i..j]$ denotes the substring of $T$ starting at position $i$ and ending at position $j$. For convenience, let $T[i..j] = \varepsilon$ for any $i' > j'$. A positive integer $p$ is said to be a period of a string $T$ if $T[i] = T[i + p]$ for all $1 \leq i \leq |T| - p$. For strings $X$ and $Y$, let $lcp(X,Y)$ denotes the length of the longest common prefix (in short, lcp) of $X$ and $Y$, i.e., $lcp(X,Y) = \max\{\ell \mid X[1..\ell] = Y[1..\ell]\}$. For a string $T$ and two integers $1 \leq i \leq j \leq |T|$, let $lce_T(i,j)$ denotes the length of the longest common extension (in short, lce) of $i$ and $j$ in $T$, i.e., $lce_T(i,j) = lcp(T[i..|T|], T[j..|T|])$. For non-empty strings $T$ and $w$, $beg_T(w)$ denotes the set of beginning positions of occurrences of $w$ in $T$. Also, for a text position $i$ in $T$, $inbeg_T(i,w)$ denotes the set of beginning positions of occurrences of $w$ in $T$ where each occurrence covers position $i$. Namely, $beg_T(w) = \{b \mid T[b..e] = w\}$ and $inbeg_T,i(w) = \{b \mid T[b..e] = w \text{ and } i \in [b,e]\}$. Further, let $xbeg_T,i(w) = beg_T(w) \setminus inbeg_T,i(w)$. For convenience, $|beg_T(\varepsilon)| = |inbeg_T,i(\varepsilon)| = |xbeg_T,i(\varepsilon)| = |T| + 1$ for any $T$ and $i$. We say that $w$ is unique in $T$ if $|beg_T(w)| = 1$, and that $w$ is repeating in $T$ if $|beg_T(w)| \geq 2$. Note that the empty string is repeating in any other string. Since every unique substring $u = T[i..j]$ of $T$ occurs exactly once in $T$, we will sometimes identify $u$ with its corresponding interval $[i,j]$. In what follows, we consider an arbitrarily fixed string $T$ of length $n \geq 1$ over an alphabet $\Sigma$ of size $\sigma = n^{O(1)}$. 


Palindromes.

For a string $w$, $w^R$ denotes the reversed string of $w$. A string $w$ is called a palindromic suffix (resp. palindromic prefix) if $w = w^R$. A palindromic suffix of a non-negative integer $n$ is a palindrome. Also, the progression difference for $s_i$ and is denoted by $p_i = |s_i - s_{i+1}|$. For a string $w$, its length-[floor of $|w|/2$] prefix (resp. length-[ceil of $|w|/2$] suffix) is called the left arm (resp. right arm) of $w$, and is denoted by $larm_w$ (resp. $rarm_w$). Also, we call $Larm_w = larm_w \cdot s_w$ (resp. $Rarm_w = s_w \cdot rarm_w$) the extended left arm (resp. extended right arm) of $w$ where $s_w$ is the character at the center of $w$ if $w$ is an odd-palindrome, and $s_w$ is empty otherwise. Note that when $w$ is an even-palindrome, $Rarm_w = rarm_w$ and $Larm_w = larm_w$. For a non-empty palindromic substring $w = T[i..j]$ of a string $T$, the center of $w$ is $\frac{i+j}{2}$ and is denoted by $center(w)$. A non-empty palindromic substring $T[i..j]$ of a string $T$ is said to be maximal if $i = 1$, $j = n$, or $T[i-1] \neq T[j+1]$. For a non-empty palindromic substring $w = T[i..j]$ of a string $T$ and a non-negative integer $\ell$, $v = T[i-\ell..j+\ell]$ is said to be an expansion of $w$ if $1 \leq i-\ell \leq j+\ell \leq n$ and $v$ is a palindrome. Also, $T[i+\ell..j-\ell]$ is said to be a contraction of $w$. Now we show periodic properties of palindromic suffixes of $T[1..i]$. Let $S_i = \{s_1, \ldots, s_g\}$ be the set of lengths of palindromic suffixes of $T[1..i]$, where $g$ is the number of palindromic suffixes of $T[1..i]$ and $s_{k-1} \leq s_k$ for $2 \leq k \leq g$. Let $p_k$ be the progression difference for $s_k$, i.e., $p_k = s_k - s_{k-1}$ for $2 \leq k \leq g$. For convenience, let $p_1 = 1$.

Then, the following results are known:

**Lemma 1** ([6] [10] [22]).

(A) For any $1 \leq k < g$, $p_{k+1} \geq p_k$.

(B) For any $1 < k < g$, if $p_{k+1} \neq p_k$, then $p_{k+1} \geq p_k + p_{k-1}$.

(C) $S_i$ can be represented by $O(\log i)$ arithmetic progressions, where each arithmetic progression is a tuple $(a, p, f)$ representing the sequence $a, a + p, \ldots, a + (f-1)p$ of lengths of $f$ palindromic suffixes with common difference $p$.

(D) The common difference $p$ is a smallest period of every palindromic suffixes of $T[1..i]$ whose length belongs to the arithmetic progression $(a, p, f)$.

For each $1 \leq j \leq f$, we will use the convention that $a(j) = a + (j-1)p$, namely $a(j)$ denotes the $j$-th shortest element for $(a, p, f)$. For simplicity, let $Y = T[1..i]$ and $Z = T[i+1..n]$. Let $lcp(x, y)$ for strings $x$ and $y$ denote the length of the longest common prefix of $x$ and $y$. Also, let $Ext(a(j))$ denote the length of the maximal palindrome that is obtained by extending $a(j)$ in $YZ$.

**Lemma 2** ([22]). For any $(a, p, f)$ of palindromic suffixes of $T[1..i]$, there exist palindromes $u, v$ and a non-negative integer $q$, such that $(w)^{f+q-1}u$ (resp. $(w)^{q}u$) is the longest (resp. shortest) palindromic suffixes represented by $(a, p, q)$ with $|w| = p$. Let $\alpha = lcp((Y[1..|Y| - a(1)])^R, Z)$ and $\beta = lcp((Y[1..|Y| - a(f)])^R, Z)$. Then $Ext(a(j)) = a(j) + 2 \min(\alpha, \beta + (f-j)p)$. Further, if there exists $a(h) \in \{a, p, f\}$ such that $a(h) + \alpha = a(f) + \beta$, then $Ext(a(h)) = a(h) + 2lcp((Y[1..|Y| - a(h)])^R, Z) \geq Ext(a(j))$ for any $j \neq h$.

By Lemmas 1 and 2 we obtain the following corollary:

**Corollary 1.** For a position $i$, divide the set of palindromic suffixes of $T[1..i]$ into groups $G_1, G_2, \ldots, G_m$, w.r.t their smallest periods. Then, $m_i \in O(\log i)$ holds. Also, for each group $G_k$, the following properties hold:

(1) The difference between centers of any two palindromes in $G_k$ is an integer power of $0.5p_k$, where $p_k$ is their smallest period.

(2) For all maximal palindromes $e_1, \ldots, e_t$ in $T$ that are expansions of palindromes in $G_k$, excluding at most one, their smallest period is also $p_k$.

(3) Among $e_1, \ldots, e_t$, any palindrome is contained by the longest one or the second longest one.

We remark that symmetric arguments hold for palindromic prefixes as well.

A non-empty string $w$ is called a 1-mismatch palindrome if there is exactly one mismatched position between $w[1..\lfloor|w|/2\rfloor]$ and $w[\lfloor|w|/2\rfloor+1..|w|^R]$. Informally, a 1-mismatch palindrome is a pseudo palindrome.
with a mismatch position between their arms. As in the case of palindromes, a 1-mismatch palindromic substring \( T[i..j] \) of a string \( T \) is said to be maximal if \( i = 1, j = n \) or \( T[i−1] \neq T[j+1] \).

A palindromic substring \( T[i..j] \) of a string \( T \) is called a minimal unique palindromic substring (MUPS) of \( T \) if \( T[i..j] \) is unique in \( T \) and \( T[i+1..j−1] \) is repeating in \( T \). We denote by \( \text{MUPS}(T) \) the set of intervals corresponding to MUPSs of a string \( T \). A MUPS cannot be a substring of another palindrome with a different center. Also, it is known that the number of MUPSs of \( T \) is at most \( n \), and set \( \text{MUPS}(T) \) can be computed in \( O(n) \) time for a given string \( T \) over an integer alphabet \( [19] \). The following lemma states that the total sum of occurrences of strings which are extended arms of MUPSs is \( O(n) \):

**Lemma 3.** The total sum of occurrences of the extended right arms of all MUPSs in a string \( T \) is at most \( 2n \). Similarly, the total sum of occurrences of the extended left arms of all MUPSs in \( T \) is at most \( 2n \).

**Proof.** It suffices to prove the former statement for the extended right arms since the latter can be proved symmetrically. Let \( w_1 \) and \( w_2 \) be distinct odd-length MUPSs of \( T \) with \( |w_1| \leq |w_2| \). For the sake of contradiction, we assume that \( T[j..j+|Rarm_{w_1}|-1] = Rarm_{w_1} \) and \( T[j..j+|Rarm_{w_2}|-1] = Rarm_{w_2} \) for some position \( j \) in \( T \). Namely, \( Rarm_{w_1} \) is a prefix of \( Rarm_{w_2} \). Then, \( larm_{w_1} \) is a suffix of \( larm_{w_2} \) by palindromic symmetry. This means that \( w_1 \) is a substring of \( w_2 \). This contradicts that \( w_2 \) is a MUPS of \( T \). Thus, all occurrences of the extended right arms of all odd-length MUPSs are different, i.e., the total number of the occurrences is at most \( n \). Similarly, the total number of all occurrences of the right arms of all even-length MUPSs is also at most \( n \). \( \square \)

### 2.2 Tools

This subsection lists some data structures used in our algorithm. Our model of computation is a standard word RAM model with machine word size \( \Omega(\log n) \).

**Suffix Trees.**

The suffix tree of \( T \) is the compacted trie for all suffixes of \( T \) \([20]\). We denote by \( \text{STree}(T) \) the suffix tree of \( T \). If a given string \( T \) is over an integer alphabet of size \( n^{O(1)} \), \( \text{STree}(T) \) can be constructed in \( O(n) \) time \([12]\). Not all substrings of \( T \) correspond to nodes in \( \text{STree}(T) \). However, the loci of such substrings can be made explicit in linear time:

**Lemma 4 (Corollary 8.1 in [20]).** Given \( m \) substrings of \( T \), represented by intervals in \( T \), we can compute the locus of each substring in \( \text{STree}(T) \) in \( O(n+m) \) total time. Moreover, the loci of all the substrings in \( \text{STree}(T) \) can be made explicit in \( O(n+m) \) extra time.

Also, this lemma implies the following corollary:

**Corollary 2.** Given \( m \) substrings of \( T \), represented by intervals in \( T \), we can sort them in \( O(n+m) \) time.

**LCE Queries.**

An LCE query on a string \( T \) is, given two indices \( i, j \) of \( T \), to compute \( \text{lce}_T(i, j) \). Using \( \text{STree}(T\$) \) enhanced with a lowest common ancestor data structure, we can answer any LCE query on \( T \) in constant time where \( \$ \) is a special character with \( \$ \notin \Sigma \). In the same way, we can compute the lcp value between any two suffixes of \( T \) or \( T^R \) in constant time by using \( \text{STree}(T\$T^R\#) \) where \( \# \) is another special character with \( \# \notin \Sigma \).

**NCA Queries.**

A nearest colored ancestor query (NCA query) on a tree \( T \) with colored nodes is, given a query node \( v \) and a color \( C \), to compute the nearest ancestor \( u \) of \( v \) such that the color of \( u \) is \( C \). Noticing that the notion of NCA is a generalization of well-known nearest marked ancestor. For NCA queries, we will use the following known results:

**Lemma 5 (\([13]\)).** Given a tree \( T \) with colored nodes, a data structure of size \( O(N) \) can be constructed in deterministic \( O(N \log \log N) \) time or expected \( O(N) \) time to answer any NCA query in \( O(\log \log N) \) time, where \( N \) is the number of nodes of \( T \).
Lemma 6 ([8, 10]). If the number of colors is O(\log N), a data structure of size O(N) can be constructed in O(N) time to answer any NCA query in O(1) time.

Eertrees.

The eertree (a.k.a. palindromic tree) of T is a pair of rooted edge-labeled trees T_{odd} and T_{even} representing all distinct palindromes in T [24]. The roots of T_{odd} and T_{even} represent ε. Each non-root node of T_{odd} (resp. T_{even}) represents an odd-palindrome (resp. even-palindrome) which occurs in T. Let pal(v) be the palindrome represented by a node v. For the root r_{odd} of T_{odd}, there is an edge (r_{odd}, u) labeled by a ∈ Σ if there is a node u with pal(u) = a. For any node v in the eertree except for r_{odd}, there is an edge (v, w) labeled by a ∈ Σ if there is a node w with pal(w) = a · pal(v) · a. We denote by EERTREE(T) the eertree of T. We will sometimes identify a node u in EERTREE(T) with its corresponding palindrome pal(u). Also, the path from a node u to a node v in EERTREE(T) is denoted by pal(u) ↼ pal(v). If a given string T is over an integer alphabet of size n^{O(1)}, EERTREE(T) can be constructed in O(n) time [24].

Path-Tree LCE Queries.

A path-tree LCE query is a generalized LCE query on a rooted edge-labeled tree T [7]: Given three nodes u, v, and w in T where u is an ancestor of v, to compute the lcp between the path-string from u to v and any path-string from w to a descendant leaf. The following result is known:

Theorem 1 (Theorem 2 of [7]). For a tree T with N nodes, a data structure of size O(N) can be constructed in O(N) time to answer any path–tree LCE query in O((\log \log N)^2) time.

We will use later path-tree LCE queries on the eertree of the input string.

Stabbing Queries.

Let I be a set of n intervals, each of which is a subinterval of the universe U = [1, O(n)]. An interval stabbing query on I is, given a query point q ∈ U, to report all intervals I ∈ I such that I is stabbed by q, i.e., q ∈ I. We can answer such a query in O(1 + k) time after O(n)-time preprocessing, where k is the number of intervals to report [26].

3 Changes of MUPSs After Single Character Substitution

In the following, we fix the original string T of length n, the text position i in T to be substituted, and the string T' after the substitution. Namely, T[i] ≠ T'[i] and T[j] = T'[j] for each j with 1 ≤ j ≤ n and j ≠ i. This section analyzes the changes of the set of MUPSs when T[i] is substituted by T'[i]. For palindromes covering editing position i, Lemma 7 holds.

Lemma 7. For a palindrome w, if inbeg_{T,i}(w) ≠ ∅, then inbeg_{T',i}(w) = ∅.

Proof. For the sake of contradiction, we assume that there is a palindrome w with inbeg_{T,i}(w) ≠ ∅ and inbeg_{T',i}(w) ≠ ∅. Let c (resp. c') be the center of an occurrence of w in T (resp. in T') covering position i. It is clear that c ≠ c' since T[i] is substituted by another character T'[i]. Also, it suffices to consider when c < c' from the symmetry of T and T'. Let d (resp. d') be the distance between c and i (resp. c' and i), i.e., d = |i − c| and d' = |i − c'|.

There are the following two cases: either (1) i ∉ [c, c'] or (2) i ∈ [c, c']. See also Fig. 1 for illustration.

1. Now we consider the case where c' < i. Another case (i < c) can be treated similarly. On the one hand, since c and c' are the centers of w, T'[c' + d] = T[c + d] = T[i]. Further, T'[c' − d] = T[c' + d] by palindromic symmetry, and hence, T'[c' − d] = T[i]. On the other hand, again, since c and c' are the centers of w, T[c − d'] = T[c' − d'] = T[i]. Further, T'[c' − d'] = T'[c' + d'] = T'[i] by palindromic symmetry, and hence, T'[c − d'] = T'[i]. Also, T'[c − d'] = T[c − d'] = T'[i] since c − d' ≠ i. Since c' − d = c − d' holds in this case, T'[i] = T'[c' − d] = T'[c − d'] = T'[i], a contradiction.

2. Similar to the first case, it can be seen that T'[c' − d] = T[c − d] = T[i]. If d = d', then T'[i] = T'[c' − d] = T'[c − d'] = T'[i], a contradiction. Hence d ≠ d' holds, and thus, T'[c + d'] = T[c + d'].

5
Lemma 8. For any position $i$ in $T$, $|W_i| \in O(\log n)$.

Proof. First, by minimality of palindromes in $W_i$, centers of palindromes in $W_i$ are different from each other. Let $W^L_i \subset W_i$ (resp. $W^R_i \subset W_i$) be the set of palindromes in $W_i$ whose center is at most $i$ (resp. at least $i$).

Let us consider the size of $W^L_i$. Every palindrome in $W^L_i$ is an expansion of some palindromic suffix of $T[1..i]$. From Corollary 1, the set of palindromic suffixes of $T[1..i]$ can be divided into $m_i \in O(\log i)$ groups w.r.t. their smallest period. Let $G_1, G_2, \ldots, G_{m_i}$ be such groups for palindromic suffixes of $T[1..i]$, and let $p_k$ be the smallest period corresponding to $G_k$. Also, for each $k$ with $1 \leq k \leq m_i$, let $E_k$ be the union set of all expansions of every palindrome in $G_k$. Let $H_k = W^L_i \cap E_k$. Since $|W^L_i| = |\bigcup_{k=1}^{m_i} H_k| = \sum_{k=1}^{m_i} |H_k|$ and $m_i \in O(\log n)$, it suffices to show that $|H_k|$ is at most a constant.

For the sake of contradiction, we assume $|H_k| \geq 4$. From [2] of Corollary 1, at least three palindromes in $H_k$ has the same smallest period $p_k$. Also, by [1] of Corollary 1, the difference between centers of any two palindromes in $H_k$ is a power of $0.5p_k$. Therefore, at least two distinct palindromes in $H_k$ have a difference of power of $p_k$ in their center positions. Let $w_1, w_2$ be such palindromes, and assume that $|w_1| \leq |w_2|$ w.l.o.g.. Then, the string between the centers of $w_1$ and $w_2$ can be represented by $x^j$ for positive integer $j$ and string of length $p_k$. Since the smallest period of $w_1$ is $p_k$, its extended right arm $R_{w_1}$ can be written by $R_{w_1} = x^{j_1} x'_1$ where $j_1$ is a non-negative integer and $x'_1$ is a proper prefix of $x$. Similarly, the extended right arm $R_{w_2}$ of $w_2$ can be written by $R_{w_2} = x^{j_2} x'_2$ where $j_2$ is a non-negative integer and $x'_2$ is a proper prefix of $x$. See also Fig. 2 for illustration. If $|w_1| = |w_2|$, then this leads $j_1 = j_2$ and $x'_1 = x'_2$, i.e., $w_1 = w_2$, a contradiction. If $|w_1| < |w_2|$, then $j_1 < j_2$ or $j_1 = j_2$ and $|x'_1| < |x'_2|$. In both cases, $R_{w_1}$ is a proper prefix of $R_{w_2}$, i.e., $w_1$ is a contraction of $w_2$. This contradicts the minimality of $w_2$. Thus $|H_k| \leq 3$ holds, and hence, we obtain $|W^L_i| \leq \sum_{k=1}^{m_i} |H_k| \leq 3m_i \in O(\log n)$.

Similarly, the size of $W^R_i$ is also $O(\log n)$. Therefore, $|W_i| \in O(\log n)$. 

Lemma 9. For each position $i$ in $T$, the number of MUPSs covering $i$ is $O(\log n)$.
Proof. By symmetry, it suffices to show that the number of MUPSs covering \( i \) and centered before \( i \) is \( O(\log n) \). Each such MUPSs is an expansion of some palindromic suffix of \( T[1..i] \). Thus, similar to the proof of Lemma \( \ref{lem:palindrome_expansion} \) we consider dividing the set of palindromic suffixes of \( T[1..i] \) into \( m_i \in O(\log i) \) groups, \( G_1, G_2, \ldots, G_{m_i} \), with respect to their smallest periods. In the following, we consider MUPSs that are expansions of palindromics in an arbitrary group \( G_k \). We show that the number of such MUPSs is at most two by contradiction. We assume the contrary, i.e., there are three MUPSs that are expansions of palindromes in \( G_k \). By Corollary \( \ref{cor:palindrome_expansion} \), at least one of the three MUPSs is a substring of an expansion of a palindrome in \( G_k \) with a different center. This contradicts that a MUPS cannot be a substring of another palindrome with a different center. Thus, the number of MUPSs that are expansions of palindromes in \( G_k \) is at most two, and we finish the proof. \( \square \)

By using Lemmas \ref{lem:palindrome_expansion} and \ref{lem:substring} we show the following theorem:

**Theorem 2.** \( |\text{MUPS}(T) \triangle \text{MUPS}(T')| \in O(\log n) \) always holds.

**Proof.** In the following, we consider the number of MUPSs to be removed.

First, at most one interval can be a MUPS of \( T \) centered at \( i \). Also, any other interval in \( \text{MUPS}(T) \) with position \( i \) cannot be an element of \( \text{MUPS}(T') \) since its corresponding string in \( T' \) is no longer a palindrome. By Lemma \( \ref{lem:substring} \) the number of such MUPSs is \( O(\log n) \).

Next, let us consider MUPSs not covering position \( i \). When a MUPS \( w \) of \( T \) not covering \( i \) is no longer a MUPS of \( T' \), then either (A) \( w \) is repeating in \( T' \) or (B) \( w \) is unique in \( T' \) but is not minimal.

Let \( w_1 \) be a MUPS of the case (A). Since \( w_1 \) does not cover \( i \), is unique in \( T \), and is repeating in \( T' \), \( |\text{inbeg}_{T,i}(w_1)| \geq 1 \) and \( |\text{xbeg}_{T,i}(w_1)| = 1 \). Let \( v_1 \) be the minimal contraction of \( w_1 \) such that \( |\text{inbeg}_{T,i}(v_1)| \geq 1 \) and \( |\text{xbeg}_{T,i}(v_1)| = 1 \). Contrary, \( w_1 \) is the only MUPS of the case (A) which is an expansion of \( v_1 \) since \( |\text{xbeg}_{T,i}(v_1)| = 1 \). Namely, there is a one-to-one relation between \( w_1 \) and \( v_1 \). By Lemma \( \ref{lem:substring} \) the number of palindromes that satisfy the above conditions of \( v_1 \) is \( O(\log n) \). Thus, the number of MUPSs of the case (A) is also \( O(\log n) \).

Let \( w_2 \) be a MUPS of the case (B). In \( T' \), \( w_2 \) covers some MUPS as a proper substring since it is not a MUPS and is unique in \( T' \). Let \( v_2 \) be the MUPS of \( T' \), which is a proper substring of \( w_2 \). While \( v_2 \) is unique in \( T' \), it is repeating in \( T \) since \( w_2 \) is a MUPS of \( T \). Namely, \( |\text{inbeg}_{T,i}(v_2)| \geq 1 \) and \( |\text{xbeg}_{T,i}(v_2)| = 1 \) hold. Also, \( v_2 \) is actually minimal: Let \( u_2 = v_2[2..|v_2| - 1] \). If we assume that \( |\text{inbeg}_{T,i}(u_2)| \geq 1 \) and \( |\text{xbeg}_{T,i}(u_2)| = 1 \), then \( u_2 \) becomes unique in \( T' \), and this contradicts that \( v_2 \) is a MUPS of \( T' \). Furthermore, similar to the above discussions, there is a one-to-one relation between \( w_2 \) and \( v_2 \). Again by Lemma \( \ref{lem:substring} \), the number of palindromes that satisfy the above conditions of \( v_2 \) is \( O(\log n) \). Thus, the number of MUPSs of the case (B) is also \( O(\log n) \).

Therefore, \( |\text{MUPS}(T) \setminus \text{MUPS}(T')| \in O(\log n) \) holds. Also, \( |\text{MUPS}(T') \setminus \text{MUPS}(T)| \in O(\log n) \) holds by symmetry.

To summarize, \( |\text{MUPS}(T) \triangle \text{MUPS}(T')| = |\text{MUPS}(T) \setminus \text{MUPS}(T')| + |\text{MUPS}(T') \setminus \text{MUPS}(T)| = |\text{MUPS}(T) \setminus \text{MUPS}(T')| + |\text{MUPS}(T') \setminus \text{MUPS}(T)| \in O(\log n) \). \( \square \)

## 4 Algorithms for Updating Set of MUPSs

In this section, we propose an algorithm for updating the set of MUPSs when a single-character in the original string is substituted by another character. We denote by \( \text{sub}(i, s) \) the substitution query, that is, to substitute \( T[i] \) by another character \( s \). First, we define a sub-problem that will be used in our algorithm:

**Problem 1.** Given a substitution query \( \text{sub}(i, s) \) on \( T \), compute the longest odd-palindromic substring \( v \) of \( T' \) such that center(\( v \)) = \( i \) and \( v \) occurs in \( T \) if it exists. Also, if such \( v \) exists, determine whether \( v \) is unique in \( T \) or not. Furthermore, if \( v \) is unique in \( T \), compute the contraction of \( v \) that is a MUPS of \( T \).

We show the following lemma:

**Lemma 10.** After \( O(n) \)-time preprocessing, we can answer Problem \( \ref{prob:substitution} \) in \( O(\delta(n, \sigma) + (\log \log n)^2) \) time where \( \delta(n, \sigma) \) denote the time to retrieve any child of the root of the odd-tree of \( \text{EERTREE}(T) \).
Proof. In the preprocessing, we construct EERTREE($T$) and apply the preprocessing for the path-tree LCE queries to the odd-tree $T_{odd}$ of EERTREE($T$). Also, we mark the nodes in EERTREE($T$) that correspond to MUPSs of $T$ and apply the preprocessing for the nearest marked ancestor (NMA) queries. The preprocessing time is $O(n)$.

Given a query $sub(i, s)$, we query the path-tree LCE between path $T[i] \leadsto larm_wT[i]rarm_w$ and tree rooted at $s$ on $T_{odd}$ where $w$ is the maximal palindrome in $T'$ centered at $i$. Let $\ell_w$ be the depth of the LCE nodes. Then the contraction $v$ of $w$ with $|rarm_v| = \ell_w$ occurs in $T$. Also, $v$ is the longest since any other contraction $u$ of $w$ longer than $v$ does not occur in $T$. Further, we can determine whether $v$ is unique in $T$ or not by checking the existence of a mark on path $s \leadsto v$. It can be done by querying NMA, and the MUPS of $T$ contained in $v$ can be computed simultaneously, if $v$ is unique in $T$.

We can compute the value $\ell_w$ in $\delta(n, \sigma)$ time for searching for the node $s$ in $T_{odd}$, plus $O((\log \log n)^2)$ time for the path-tree LCE query.

When $\sigma \in O(n)$, we can easily achieve $\delta(n, \sigma) \in O(1)$ with linear space, by using an array of size $\sigma$. Otherwise, we achieve $\delta(n, \sigma) \in O(\log \sigma)$ for a general ordered alphabet by using a binary search tree.

In the rest of this paper, we propose an algorithm to compute the changes in MUPSs after a single-character substitution. We compute MUPSs to be removed and added separately. We show how to compute all MUPSs to be removed in Section 4.1. Also, we show how to compute all MUPSs to be added in Section 4.2. In Section 4.3, we introduce another solution for Problem 1. Our strategy is basically to pre-compute changes in MUPSs for some queries as much as possible within linear time. The other changes will be detected on the fly by using some data structures.

4.1 Computing MUPSs to Be Removed

We categorize MUPSs to be removed into three types as follows:

R1) A MUPS of $T$ that covers $i$.

R2) A MUPS of $T$ that does not cover $i$ and is repeating in $T'$.

R3) A MUPS of $T$ that does not cover $i$ and is unique but not minimal in $T'$.

See Fig. 3 for illustration. In the following, we describe how to compute all MUPSs for each type separately.

Type R1.

All MUPSs covering editing position $i$ are always removed. Thus, we can detect them in $O(1 + \alpha_{rem})$ time after a simple linear time preprocessing (e.g., using stabbing queries), where $\alpha_{rem}$ is the number of MUPSs of Type R1.

Type R2.

Before describing our algorithm, we give a few observations about MUPSs of Type R2. Let $w$ be a MUPS of Type R2. Since $w$ is unique in $T$ and is repeating in $T'$, $|inbeg_{T',w}(w)| \geq 1$. When $w$ occurs in $T'$ centered at editing position $i$, we retrieve such $w$ by applying Problem 1. If it is not the case, we can utilize the following observations: Consider the starting position $j$ of an occurrence of $w$ in $T'$ such that $T'[j..j + |w| - 1] = w$ and $i \in [j..j + |w| - 1]$. If position $i$ is covered in the right arm of $T'[j..j + |w| - 1]$, then $Larm_w$ occurs at position $j$ in both $T$ and $T'$. Further, the Hamming distance between $T[j + |Larm_w|..j + |w| - 1]$ and $w||Larm_w| + 1..|w| = rarm_w$ equals 1. Namely, for each occurrence at position $k$ of string $Larm_w$ in $T$, $w$ can occur at $k$ in $T'$ only if the Hamming distance between $T[k + |Larm_w|..k + |w| - 1]$ and $w||Larm_w| + 1..|w|| equals 1. In other words, if the Hamming distance is greater than 1, $w$ cannot occur at $k$ in $T'$. See also Fig. 4. The strategy of the algorithm for MUPSs of Type R2 is following: A MUPS of Type R2 having occurrence centered at editing position $i$ in $T'$ is found by applying Problem 1. For the remaining MUPSs of Type R2, we precompute them by using the above observations.
Preprocessing. In the preprocessing phase, we first apply the $O(n)$-time preprocessing of Lemma 10 for Problem 1. Next, we initialize the set $A_{R2} = \emptyset$. The set $A_{R2}$ will become an index of MUPSs of Type R2 when the preprocessing is finished. For each MUPS $w = T[b..e]$ of $T$, we process the followings: For the beginning position $j \neq b$ of each occurrence of $\text{Larm}_w$ in $T$, we first compute the lcp value between $T[j + |\text{Larm}_w|..|T|]$ and $\text{rarm}_w$ with allowing one mismatch. Note that $T[j + |\text{Larm}_w|..|T|]$ must have at least one mismatch with $\text{rarm}_w$, since $T[j..j + |\text{Larm}_w| - 1] = \text{Larm}_w$, $j \neq b$, and $T[b..e]$ is unique in $T$. If there are two mismatch positions between them, do nothing for this occurrence since $w$ cannot occur at $j$ after any substitution. We can check this by querying LCE at most twice. Otherwise, let $q = j + |\text{Larm}_w| - 1 + d$ be the mismatched position in $T$. When the $q$-th character of $T$ is substituted by the character $\text{rarm}_w[d]$, $w = T[b..e]$ occurs at $j \neq b$, i.e., it is a MUPS of Type R2. So we add MUPS $w = T[b..e]$ into $A_{R2}$ with the pair of index and character $(q, \text{rarm}_w[d])$ as the key. In addition, symmetrically, we update $A_{R2}$ for each occurrence of $\text{Rarm}_w$ in $T$. After finishing the above processes for every MUPS of $T$, we sort the elements of $A_{R2}$ by radix sort on the keys. If there are multiple identical elements with the same key, we unify them into a single element. Also, if there are multiple elements with the same key, we store them in a linear list. By Lemma 3, the total number of occurrences of arms of MUPSs is $O(n)$, and hence, the total preprocessing time is $O(n)$.

Query. Given a query $\text{sub}(i, s)$, we query Problem 1 with the same pair $(i, s)$ as the input. Then, we complete checking whether there exists a MUPS of Type R2 centered at $i$. Next, consider the existence
of the remaining MUPSs of Type R2. First, an element in $A_{R2}$ corresponding to the key $(i, s)$ can be detected in $O(\log \sigma_i)$ time by using random access on indices and binary search on characters, where $\sigma_i$ is the number of characters $s_i$ such that the key $(i, s_i)$ exists in $A_{R2}$. After that, we can enumerate all the other elements with the key by scanning the corresponding linear list. Thus, the total query time is $O(\delta(n, \sigma) + (\log \log n)^2 + \log \sigma_i + \beta_{\text{rem}})$ where $\beta_{\text{rem}}$ is the number of MUPSs of Type R2. Finally, we show $\sigma_i \in O(\min\{\sigma, \log n\})$. Let us consider palindromes in $T'$ whose right arm covers position $i$. Those whose left arms cover $i$ can be treated similarly. Any palindrome in $T'$ whose right arm covers $i$ is an expansion of some maximal palindrome in $T$ ending at $i - 1$. It is known that the number of possible characters immediately preceding such maximal palindromes is $O(\log n)$ [15]. Therefore, $\sigma_i \in O(\log n)$ holds, and thus, the query time is $O(\delta(n, \sigma) + (\log \log n)^2 + \beta_{\text{rem}})$.

Type R3.

Let $w = T[b, e]$ be a MUPS of $T$ and let $v = T[b + 1..e - 1]$. Further let $T[b_{r1}, e_{l1}]$ and $T[b_{r1}, e_{r1}]$ be the leftmost and the rightmost occurrence of $v$ in $T$ except for $T[b + 1..e - 1]$. We define interval $\rho_w = \{k \mid k \not\in [b + 1, e - 1] \text{ and } k \in [b_{r1}, e_{l1}]\}$. Note that $\rho_w$ can be empty. See also Fig. 5 for illustration. If the editing position $i$ is in $\rho_w$, then the only occurrence of $v$ in $T'$ is $T'[b + 1..e - 1]$, i.e., $v$ is unique in $T'$. Thus, $w$ is a removed MUPS of Type R3. Contrary, if $i \not\in [b, e]$ and $i \not\in \rho_w$, there are at least two occurrences of $v$ in $T'$, i.e., $w$ cannot be a MUPS of Type R3.

**Preprocessing.** First, we compute the set of intervals $\mathcal{R} = \{\rho_w \mid w \text{ is a MUPS of } T\}$. $\mathcal{R}$ can be computed by traversing over the suffix tree of $T$ enhanced with additional explicit nodes, each of which represents a substring $T[b + 1..e - 1]$ for each MUPS $T[b, e]$ of $T$. Also, we apply the preprocessing for stabbing queries to $\mathcal{R}$. The total time for preprocessing is $O(n)$.

**Query.** Given a query $\text{sub}(i, s)$, compute all intervals in $\mathcal{R}$ stabbed by position $i$ by answering a stabbing query. They correspond to MUPSs of Type R3. The query time is $O(1 + \gamma_{\text{rem}})$, where $\gamma_{\text{rem}}$ is the number of MUPSs of Type R3.

To summarize, we can compute all MUPSs to be removed after a single-character substitution in $O(\delta(n, \sigma) + (\log \log n)^2 + \alpha_{\text{rem}} + \beta_{\text{rem}} + \gamma_{\text{rem}})$ time.

### 4.2 Computing MUPSs to be Added

Next, we propose an algorithm to detect MUPSs to be added after a substitution. As in Section 4.1, we categorize MUPSs to be added into three types:

**A1)** A MUPS of $T'$ that covers $i$.

**A2)** A MUPS of $T'$ that does not cover $i$ and is repeating in $T$.

**A3)** A MUPS of $T'$ that does not cover $i$ and is unique but not minimal in $T$.
Figure 6: Illustration for four types of MUPSs to be added. The red arrows represent MUPSs. $w_{1-1}$, $w_{1-2}$, $w_2$, and $w_3$ are MUPSs of Type A1-1, Type A1-2, Type A2, and Type A3 in $T'$, respectively. Also, $v$ is the MUPS of $T$ that is a contraction of $w_3$. It is unique in $T$, but is not unique in $T'$. Furthermore, we categorize MUPSs of Type A1 into two sub-types:

A1-1) A MUPS of $T'$ that covers position $i$ in its arm.

A1-2) A MUPS of $T'$ centered at editing position $i$.

See Fig. 6 for illustration.

Type A1-1.

A MUPS of Type A1-1 is a contraction of some maximal palindrome in $T'$ covering editing position $i$ in its arm. Further, such a maximal palindrome in $T'$ corresponds to some 1-mismatch maximal palindrome in $T$, which covers $i$ as a mismatch position. Thus, we preprocess for arms of each 1-mismatch maximal palindrome in $T$. For MUPSs of Type A1, we utilize the following observation:

Observation 1. For any palindrome $v$ covering position $i$ in $T'$, $v$ is unique in $T'$ if and only if $|\text{inbeg}_{T',i}(v)| = 1$ and $|\text{xbeg}_{T',i}(v)| = 0$.

Preprocessing. In the preprocessing phase, we first consider sorting extended arms of 1-mismatch maximal palindromes in $T$. Let $EA$ be the multiset of strings that consists of the extended right arms and the reverse of the extended left arms of all 1-mismatch maximal palindromes in $T$. Note that each string in $EA$ can be represented in constant space since it is a substring of $T$ or $TR$. Let $MA'$ be a lexicographically sorted array of all elements in $EA$. Here, the order between the same strings can be arbitrary. Also, for each string in $EA$, we consider a quadruple of the form $(\text{par}, \text{pos}, \text{chr}, \text{rnk})$ where $\text{par} \in \{\text{odd}, \text{even}\}$ represents the parity of the length of the corresponded 1-mismatch maximal palindrome, $\text{pos}$ is the mismatched position on the opposite arm, $\text{chr}$ is the mismatched character on the extended arm, and $\text{rnk}$ is the rank of the extended arm in $MA'$. Let $MA$ be a radix sorted array of these quadruples. It can be seen that for each triple $(p, i, s)$ of parity $p$, mismatched position $i$, and mismatched character $s$, all elements corresponding to the triple are stored continuously in $MA$. We denote by $MA_{p, i, s}$ the subarray of $MA$ consists of such elements. In other words, $MA_{p, i, s}$ is a sorted array of extended arms of maximal palindromes of parity $p$ covering position $i$ in $T'$ when the $i$-th character of $T$ is substituted by $s$. 11
Now let us focus on odd-palindromes. Even-palindromes can be treated similarly. We construct the suffix tree of $T$ and make the loci of strings in $EA$ explicit. We also make the loci of the extended right arm of every odd-palindrome in $T$ explicit. Simultaneously, we mark the nodes corresponding to the extended right arms and apply the preprocessing for the nearest marked ancestor (NMA) queries to the marked tree. We denote the tree by $ST_{odd}$. Next, we initialize the set $A_{A1,1} = \emptyset$. The set $A_{A1,1}$ will become an index of MUPSs of Type A1-1 when the preprocessing is finished. For each non-empty $MA_{odd,i,s}$ and for each string $w$ in $MA_{odd,i,s}$, we do the followings: Let $x_w$ be the odd-palindrome whose extended right arm is $w$ when $T[i]$ is substituted by $s$. Let $u$ and $v$ are the preceding and the succeeding string of $w$ in $MA_{odd,i,s}$ (if such palindromes do not exist, they are empty). Further let $\ell_w = \max\{lcp(u,w),lcp(w,v)\}$. When $T[i]$ is substituted by $s$, any contraction $y$ of $x_w$ such that $y$ covers position $i$ and the arm-length of $y$ is at least $\ell_w$ has only one occurrence which covers position $i$ in $T'$, i.e., $|inbeg_{T',i}(y)| = 1$. Next, we query the NMA for the node corresponding to $w$ on $ST_{odd}$. Let $\ell'_w$ be the length of the extended right arm obtained by the NMA query. When $T[i]$ is substituted by $s$, any contraction $y'$ of $x_w$ such that the arm-length of $y'$ is at least $\ell'_w$, has no occurrences which do not cover position $i$ in $T'$, i.e., $|xbeg_{T',i}(y')| = 0$. Thus, by Observation 1, the contraction $y^*$ of $x_w$ of arm-length $\max\{\ell_w,\ell'_w\}$ is a MUPS of Type A1-1 for the query $sub(i,s)$, if such $y^*$ exists. In such a case, we store the information about $y^*$ (i.e., its center and radius) into $A_{A1,1}$ using (odd, $i, s$) as the key. After finishing the above preprocessing for all strings in $EA$, we sort all elements in $A_{A1,1}$ by their keys.

Since each element in $EA$ is a substring of $T\#ST_{odd}^R\#$, they can be sorted in $O(n + |EA|) = O(n)$ time by Corollary 2. Namely, $MA'$ can be computed in linear time, and thus $MA$ too. By Lemma 4, tree $ST_{odd}$ can be constructed in $O(n)$ time. Also, we can answer each NMA query and LCP query in constant time after $O(n)$ time preprocessing. Hence, the total preprocessing time is $O(n)$.

**Query.** Given a query $sub(i,s)$, we compute all MUPSs of Type A1-1 by searching for elements in $A_{A1,1}$ with keys (odd, $i, s$) and (even, $i, s$). An element with each of the keys can be found in $O(\log \min\{\sigma, \log n\})$ time. Thus, all MUPSs of Type A1-1 can be computed in $O(\log \min\{\sigma, \log n\} + \alpha'_{odd})$ time where $\alpha'_{odd}$ is the number of MUPSs of Type A1-1.

**Type A1-2.**

The MUPS of Type A1-2 is a contraction of the maximal palindrome in $T'$ centered at $i$. By definition, there is at most one MUPS of Type A1-2.

**Preprocessing.** In the preprocessing phase, we again construct $MA$ and related data structures as in Type A1-1. Further, we apply the $O(n)$-time preprocessing of Lemma 10 for Problem 1. The total preprocessing time is $O(n)$.

**Query.** Given substitution query $sub(i,s)$, we compute the MUPS centered at $i$ in $T'$ as follows (if it exists): It is clear that $T'[i..i] = s$ is the MUPS of Type A1-2 if $s$ is a unique character in $T'$. In what follows, we consider the other case. Let $w$ be the maximal palindrome centered at $i$ in $T'$. First, we compute the maximum lcp value $\ell_w$ between $Rarm_w$ and extended arms in $MA_{odd,i,s}$. Then, any contraction $y$ of $w$ such that the arm-length of $y$ is at least $\ell_w$, has no occurrences which cover position $i$ in $T'$, i.e., $|inbeg_{T',i}(y)| = 1$. We can compute $\ell_w$ in $O(\log \min\{\sigma, \log n\})$ time, combining LCE queries and binary search. Note that $rarm_w$ occurs at $i + 1$ in both $T$ and $T'$ while $Rarm_w$ might be absent from $T$. Next we compute the arm-length $\ell'_w$ of the shortest palindrome $v$ such that center($v$) = $i$ and $|xbeg_{T',i}(v)| = 0$, i.e., $v$ is absent from $T$. Since the contraction $\tilde{v}$ of $w$ of arm-length $\ell'_w - 1$ is the longest palindrome such that center($\tilde{v}$) = $i$ and $\tilde{v}$ occurs in $T$, we can reduce the problem of computing $\ell'_w$ to Problem 1. Thus, we can compute $\ell'_w$ in $O(\delta(n,\sigma) + (\log \log n)^2)$ time by Lemma 10. Similar to the case of Type A1-1, by Observation 1 the contraction $y^*$ of $x_w$ of arm-length $\max\{\ell_w, \ell'_w\}$ is a MUPS of Type A1-2, if such $y^*$ exists. Therefore, the MUPS of Type A1-2 can be computed in $O(\delta(n,\sigma) + (\log \log n)^2)$ time.
Type A2.

A MUPS of Type A2 occurs at least twice in $T$, and there is only one occurrence not covering editing position $i$. For a palindrome $w$ repeating in $T$, let $T[b_{11}..e_{11}]$ and $T[b_{12}..e_{12}]$ be the leftmost and the second leftmost occurrence of $w$ in $T$. Further, let $T[b_{11}..e_{11}]$ and $T[b_{12}..e_{12}]$ be the rightmost and the second rightmost occurrence of $w$ in $T$. We define interval $\rho_w$ as the intersection of all occurrences of $w$ except for the leftmost one, i.e., $\rho_w = \{k \mid k \notin [b_{11}, e_{11}] \text{ and } k \in [b_{12}, e_{12}]\}$ (see also Fig. 7). Similarly, we define interval $\tilde{\rho}_w$ as the intersection of all occurrences of $w$ except for the rightmost one. Note that $\rho_w$ and $\tilde{\rho}_w$ can be empty. Then, $w$ is unique after the $i$-th character is edited if and only if $i \in \rho_w \cup \tilde{\rho}_w$. Thus, any MUPS of Type A2 is a palindrome corresponding to some interval in $\rho_w \cup \tilde{\rho}_w$ stabbed by $i$. To avoid accessing intervals that do not correspond to the MUPSs to be added, we decompose each $\rho_w$. It is easy to see that for any contraction $v$ of $w$, $\rho_v \subseteq \rho_w$ holds. Also, if $T[i]$, with $i \in \rho_v$, is edited, then both $w$ and $v$ become unique in $T^\prime$, i.e., $w$ cannot be a MUPS of $T^\prime$. For each unique palindrome $w$ in $T$, we decompose $\rho_w$ into at most three intervals $\rho_w = \rho_1 \cup \rho_2 \cup \rho_3$, where $\rho_1 \subseteq \rho_2 \subseteq \rho_3$ and $\rho_3$ is a removed MUPS of $T$. Then, $w$ is a MUPS of Type A2 if and only if $i \in \rho_1 \cup \rho_2 \cup \rho_3$.

Preprocessing. In the preprocessing phase, we first construct the eertree of $T$ and the suffix tree of $T$ enhanced with additional explicit nodes for all distinct palindromes in $T$. Next, we compute at most four (leftmost, second leftmost, rightmost, second rightmost) occurrences of each palindrome in $T$ by traversing the enhanced suffix tree. At the same time, we compute $\rho_w$ and $\tilde{\rho}_w$ for each palindrome in $w$. Next, we sequentially access distinct palindromes by traversing EERTREE$(T)$ in a pre-order manner. Then, for each palindrome $w$, we decompose $\rho_w$ and $\tilde{\rho}_w$ based on the rules as mentioned above. Finally, we apply the preprocessing for stabbing queries to the $O(n)$ intervals obtained. The total preprocessing time is $O(n)$.

Query. Given a query $\text{sub}(i,s)$, we compute all intervals stabbed by position $i$. The palindromes corresponding to the intervals are MUPSs of Type A2. Hence, the query time is $O(1 + \beta_{add})$, where $\beta_{add}$ is the number of MUPSs of Type A2.

Type A3.

A MUPS of Type A3 is unique but not minimal in $T$. Such a unique palindrome $u$ in $T$ contains a MUPS $w \neq u$ of $T$ as a contraction. Since $u$ is a MUPS of $T^\prime$, $w$ is repeating in $T^\prime$, i.e., $w$ is a removed MUPS of Type R2. Contrary, consider a MUPS $w$ of Type R2, which is repeating in $T^\prime$. Then, the shortest unique expansion of $w$ in $T^\prime$ is an added MUPS of Type A3, if it exists. The preprocessing for Type A3 is almost the same as for Type R2. We store a bit more information for Type A3 in addition to the information in $A_{R2}$.

Preprocessing. In the preprocessing phase, we first apply the $O(n)$-time preprocessing of Lemma 10 for Problem 1. Next, we initialize the set $A_{A3} = \emptyset$. This set $A_{A3}$ will become an index of MUPSs of Type A3 when the preprocessing is finished. For each MUPS $w = T[b..e]$ of $T$, we process the followings: For the beginning position $j \neq b$ of each occurrence of Larm$_w$ in $T$, we compute the lcp value $\ell_j$ between

---

Figure 7: Illustration for $\rho_w$ of Type A2. The top two arrows represent the leftmost and the second leftmost occurrence of $w$. Also, the bottom two arrows represent the second rightmost and the rightmost occurrence of $w$. In this case, $\rho_w = [b_{11}, e_{12}]$. 


can be done in $O$ time. The total query time of Type A3 is $O(\log \log n)$ for each occurrence of $\text{Rarm}_w$ in $T$. After finishing the above processes for every MUPS of $T$, we then sort the elements of $A_{A3}$ by radix sort on the keys. If there are multiple identical elements with the same key, we unify them into a single element. Also, if there are multiple elements with the same key, we store them in a linear list. By Lemma 3, the total number of occurrences of arms of MUPSs is $O(n)$, and hence, the total preprocessing time is $O(n)$. 

**Query.** Given a query $\text{sub}(i, s)$, we query Problem 1 with the same pair $(i, s)$ as the input. Then, we complete checking whether there exists a MUPS of Type A3 centered at $i$. For the remaining MUPSs of Type 3, we retrieve the MUPSs of Type A3 using the index $A_{A3}$ as in the query algorithm for Type R2. This can be done in $O(\log \min\{\sigma, \log n\} + \gamma_{add})$ time where $\gamma_{add}$ is the number of MUPSs of Type A3. Therefore, the total query time of Type A3 is $O(\delta(n, \sigma) + (\log \log n)^2 + \gamma_{add})$.

To summarize, we can compute all MUPSs to be added after a single-character substitution in $O(\delta(n, \sigma) + (\log \log n)^2 + \alpha'_{add} + \beta_{add} + \gamma_{add})$ time. Then, combining the results of Section 4.1, we obtain the following theorem:

**Theorem 3.** After $O(n)$-time preprocessing, we can compute the set of MUPSs after a single-character substitution in $O(\delta(n, \sigma) + (\log \log n)^2 + d) \subset O(\log n)$ time where $d$ is the number of changes of MUPSs.

### 4.3 Alternative Algorithm for Problem 1

The query time of Theorem 3 is dominated by the time to answer Problem 1. Here, we introduce another solution for Problem 1 utilizing nearest colored ancestor queries instead of path-tree LCE queries.

**Preprocessing for Problem 1** We first construct the suffix tree of $T$. Also, for each odd palindrome in $T$, we make the locus of the right arm explicit and label the node with the pair of the center character and the binary flag that indicates if the palindrome is a MUPS. We regard the pair as the color of the node. Furthermore, we apply a preprocessing for NCA queries to the colored tree. The preprocessing time is $O(n + c_{nca}(n, \sigma))$, where $c_{nca}(n, \sigma)$ is the preprocessing time for NCA queries.

**Query for Problem 1** Given a substitution query $\text{sub}(i, s)$, we start at the node corresponding to $\text{Rarm}_w$ where $w$ is the maximal palindrome in $T$ centered at $i$. We then compute the nearest ancestor $V$ colored with $(s, 0)$ by using NCA query. If such node $V$ exists, palindrome $P = \text{str}(V)^R \cdot s \cdot \text{str}(V)$ is the answer of the former part of Problem 1 where $\text{str}(V)$ denotes the string corresponding to $V$ in the enhanced suffix tree of $T$. Also, we query NCA $(s, 1)$ from $V$. We can determine if $P$ is unique, and if it is unique, we can find the MUPS contained in $P$. The query time is $O(q_{nca}(n, \sigma))$ where $q_{nca}(n, \sigma)$ is the query time for NCA.

Let $s_{nca}(n, \sigma)$ denote the space for the NCA data structure. We obtain the following theorem:

**Theorem 4.** After $O(n + c_{nca}(n, \sigma))$-time and $O(n + s_{nca}(n, \sigma))$-space preprocessing, we can compute the set of MUPSs after a single-character substitution in $O(q_{nca}(n, \sigma) + \log \min\{\sigma, \log n\} + d)$ time.

The results for NCA queries in Lemmas 5 and 6 can be plugged into the functions $c_{nca}$, $q_{nca}$, and $s_{nca}$. In addition, even when a general case, we can handle $\delta(n, \sigma)$ as a constant by utilizing a perfect hashing [12] after $O(n\log \log n)$-time or $O(n)$-expected time preprocessing. Table 1 lists different representations of the time/space complexities of Theorems 3 and 4. We emphasize that our algorithm runs in optimal $O(1 + d)$ time when $\sigma$ is constant.

---

1. There can be a node with multiple colors in the tree. However, we can easily avoid such a situation by copying a node with $k$ colors to $k$ nodes. Also, in the case of Problem 1, the cumulative total number of colored nodes is $O(n)$. 

14
Table 1: Concrete complexities of our algorithms for the problem of computing MUPSs after a single-character substitution. All the above results require only linear space. Each query time is $O(\log n)$ since $\log \sigma = O(\log \log n)$ and $d = O(\log n)$.

Corollary 3. If $\sigma = O(1)$, after $O(n)$-time and $O(n)$-preprocessing, we can compute the set of MUPSs after a single-character substitution in $O(1 + d)$ time.

5 Conclusions and Future Work

In this paper, we dealt with the problem of updating the set of MUPSs of a string after a single-character substitution. We showed that the number $d$ of changes of MUPSs after a single-character substitution is $O(\log n)$. Furthermore, we presented an algorithm that uses $O(n)$ time and space for preprocessing, and updates the set of MUPSs in $O(\log \sigma + (\log \log n)^2 + d)$ time where $\sigma$ is the alphabet size. We also proposed a variant of the algorithm, which runs in optimal $O(1 + d)$ time when the alphabet size is constant.

Our future work includes the following:

(1) Can our algorithm be adapted to the cases of insertions and deletions?

(2) Can we extend our algorithm to a fully dynamic setting? It is interesting whether the techniques in [5, 2] can be utilized in the dynamic version of the MUPS problem.

Acknowledgements

We would like to thank Associate Professor Shunsuke Inenaga (Kyushu University) for the valuable discussions on simplifying our algorithms. This work was supported by the JSPS KAKENHI Grant Numbers JP20J21147 (MF) and JP20J11983 (TM).
References

[1] Paniz Abedin, Sahar Hooshmand, Arnab Ganguly, and Sharma V. Thankachan. The heaviest induced ancestors problem revisited. In Gonzalo Navarro, David Sankoff, and Binhai Zhu, editors, Annual Symposium on Combinatorial Pattern Matching, CPM 2018, July 2-4, 2018 - Qingdao, China, volume 105 of LIPIcs, pages 20:1–20:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.CPM.2018.20

[2] Amihood Amir and Itai Boneh. Dynamic palindrome detection. CoRR, abs/1906.09732, 2019. URL: http://arxiv.org/abs/1906.09732 arXiv:1906.09732

[3] Amihood Amir, Itai Boneh, Panagiotis Charalampopoulos, and Eitan Kondratovsky. Repetition detection in a dynamic string. In Michael A. Bender, Ola Svensson, and Grzegorz Herman, editors, 27th Annual European Symposium on Algorithms, ESA 2019, September 9-11, 2019, Munich/Garching, Germany, volume 144 of LIPIcs, pages 5:1–5:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.ESA.2019.5

[4] Amihood Amir, Panagiotis Charalampopoulos, Costas S. Iliopoulos, Solon P. Pissis, and Jakub Radoszewski. Longest common factor after one edit operation. In Gabriele Fici, Marinella Sciortino, and Rossano Venturini, editors, String Processing and Information Retrieval - 24th International Symposium, SPIRE 2017, Palermo, Italy, September 26-29, 2017, Proceedings, volume 10508 of Lecture Notes in Computer Science, pages 14–26. Springer, 2017. doi:10.1007/978-3-319-67428-5_2

[5] Amihood Amir, Panagiotis Charalampopoulos, Solon P. Pissis, and Jakub Radoszewski. Longest common substring made fully dynamic. In Michael A. Bender, Ola Svensson, and Grzegorz Herman, editors, 27th Annual European Symposium on Algorithms, ESA 2019, September 9-11, 2019, Munich/Garching, Germany, volume 144 of LIPIcs, pages 6:1–6:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.ESA.2019.6

[6] Alberto Apostolico, Dany Breslauer, and Zvi Galil. Parallel detection of all palindromes in a string. Theor. Comput. Sci., 141(1&2):163–173, 1995. doi:10.1016/0304-3975(94)00083-U

[7] Philip Bille, Pawel Gawrychowski, Inge Li Gørtz, Gad M. Landau, and Oren Weimann. Longest common extensions in trees. Theor. Comput. Sci., 638:98–107, 2016. doi:10.1016/j.tcs.2015.08.009

[8] Philip Bille, Gad M. Landau, Rajeev Raman, Kunihiko Sadakane, Srinivasa Rao Satti, and Oren Weimann. Random access to grammar-compressed strings and trees. SIAM J. Comput., 44(3):513–539, 2015. doi:10.1137/130936889

[9] Panagiotis Charalampopoulos, Pawel Gawrychowski, and Karol Pokorski. Dynamic longest common substring in polylogarithmic time. In Artur Czumaj, Anuj Dawar, and Emanuela Merelli, editors, 47th International Colloquium on Automata, Languages, and Programming, ICALP 2020, July 8-11, 2020, Saarbrücken, Germany (Virtual Conference), volume 168 of LIPIcs, pages 27:1–27:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.ICALP.2020.27

[10] Panagiotis Charalampopoulos, Tomasz Kociumaka, Manal Mohamed, Jakub Radoszewski, Wojciech Rytter, and Tomasz Waleń. Internal dictionary matching. Algorithmica, 2021. doi:10.1007/s00453-021-00821-y

[11] Xavier Droubay, Jacques Justin, and Giuseppe Pirillo. Episturmian words and some constructions of de Luca and Rauzy. Theor. Comput. Sci., 255(1-2):539–553, 2001. doi:10.1016/S0304-3975(99)00320-5

[12] Martin Farach-Colton, Paolo Ferragina, and S. Muthukrishnan. On the sorting-complexity of suffix tree construction. J. ACM, 47(6):987–1011, 2000. doi:10.1145/355541.355547

[13] Michael L. Fredman, János Komlós, and Endre Szemerédi. Storing a sparse table with O(1) worst case access time. J. ACM, 31(3):538–544, 1984. doi:10.1145/828.1884
[14] Mitsuru Funakoshi and Takuya Mieno. Minimal unique palindromic substrings after single-character substitution. In Thierry Lecroq and Hélène Touzet, editors, String Processing and Information Retrieval - 28th International Symposium, SPIRE 2021, Lille, France, October 4-6, 2021, Proceedings, volume 12944 of Lecture Notes in Computer Science, pages 33–46. Springer, 2021. doi:10.1007/978-3-030-86692-1_4.

[15] Mitsuru Funakoshi, Yuto Nakashima, Shunsuke Inenaga, Hideo Bannai, and Masayuki Takeda. Computing longest palindromic substring after single-character or block-wise edits. Theor. Comput. Sci., 859:116–133, 2021. doi:10.1016/j.tcs.2021.01.014.

[16] Leszek Gasieniec, Marek Karpinski, Wojciech Plandowski, and Wojciech Rytter. Efficient algorithms for Lempel-Ziv encoding. In Rolf Karlsson and Andrzej Lingas, editors, Algorithm Theory — SWAT’96, pages 392–403, Berlin, Heidelberg, 1996. Springer Berlin Heidelberg.

[17] Paweł Gawrychowski, Tomohiro I, Shunsuke Inenaga, Dominik Köppl, and Florin Manea. Tighter bounds and optimal algorithms for all maximal α-gapped repeats and palindromes - finding all maximal α-gapped repeats and palindromes in optimal worst case time on integer alphabets. Theory Comput. Syst., 62(1):162–191, 2018. doi:10.1007/s00224-017-9794-5.

[18] Paweł Gawrychowski, Gad M. Landau, Shay Mozes, and Oren Weimann. The nearest colored node in a tree. Theor. Comput. Sci., 710:66–73, 2018. doi:10.1016/j.tcs.2017.08.021.

[19] Hiroe Inoue, Yuto Nakashima, Takuya Mieno, Shunsuke Inenaga, Hideo Bannai, and Masayuki Takeda. Algorithms and combinatorial properties on shortest unique palindromic substrings. J. Discrete Algorithms, 52-53:122–132, 2018. doi:10.1016/j.jda.2018.11.009.

[20] Tomasz Kociumaka, Marcin Kubica, Jakub Radoszewski, Wojciech Rytter, and Tomasz Walen. A linear-time algorithm for seeds computation. ACM Trans. Algorithms, 16(2):27:1–27:23, 2020. doi:10.1145/3386369.

[21] Glenn K. Manacher. A new linear-time "on-line" algorithm for finding the smallest initial palindrome of a string. J. ACM, 22(3):346–351, 1975. doi:10.1145/321892.321896.

[22] Wataru Matsubara, Shunsuke Inenaga, Akira Ishino, Ayumi Shinohara, Tomoyuki Nakamura, and Kazuo Hashimoto. Efficient algorithms to compute compressed longest common substrings and compressed palindromes. Theor. Comput. Sci., 410(8-10):900–913, 2009. doi:10.1016/j.tcs.2008.12.016.

[23] Takuya Mieno, Kiichi Watanabe, Yuto Nakashima, Shunsuke Inenaga, Hideo Bannai, and Masayuki Takeda. Palindromic trees for a sliding window and its applications. Information Processing Letters, 173:106174, 2022. doi:https://doi.org/10.1016/j.ipl.2021.106174.

[24] Mikhail Rubinchik and Arseny M. Shur. EERTREE: an efficient data structure for processing palindromes in strings. Eur. J. Comb., 68:249–265, 2018. doi:10.1016/j.ejc.2017.07.021.

[25] Mikhail Rubinchik and Arseny M. Shur. Palindromic k-factorization in pure linear time. In Javier Esparza and Daniel Král’, editors, 45th International Symposium on Mathematical Foundations of Computer Science, MFCS 2020, August 24-28, 2020, Prague, Czech Republic, volume 170 of LIPIcs, pages 81:1–81:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.MFCS.2020.81.

[26] Jens M. Schmidt. Interval stabbing problems in small integer ranges. In Yingfei Dong, Ding-Zhu Du, and Oscar H. Ibarra, editors, Algorithms and Computation, 20th International Symposium, ISAAC 2009, Honolulu, Hawaii, USA, December 16-18, 2009. Proceedings, volume 5878 of Lecture Notes in Computer Science, pages 163–172. Springer, 2009. doi:10.1007/978-3-642-10631-6_18.

[27] Yuki Urabe, Yuto Nakashima, Shunsuke Inenaga, Hideo Bannai, and Masayuki Takeda. Longest Lyndon substring after edit. In Gonzalo Navarro, David Sankoff, and Binhai Zhu, editors, Annual Symposium on Combinatorial Pattern Matching, CPM 2018, July 2-4, 2018 - Qingdao, China, volume 105 of LIPIcs,
[28] Kiichi Watanabe, Yuto Nakashima, Shunsuke Inenaga, Hideo Bannai, and Masayuki Takeda. Fast algorithms for the shortest unique palindromic substring problem on run-length encoded strings. *Theory Comput. Syst.*, 64(7):1273–1291, 2020. doi:10.1007/s00224-020-09980-x

[29] Peter Weiner. Linear pattern matching algorithms. In *14th Annual Symposium on Switching and Automata Theory, Iowa City, Iowa, USA, October 15-17, 1973*, pages 1–11. IEEE Computer Society, 1973. doi:10.1109/SWAT.1973.13