Shortest Cover After Edit

Kazuki Mitani
Graduate School of Information Science and Technology, Hokkaido University, Japan

Takuya Mieno
Department of Computer and Network Engineering, University of Electro-Communications, Tokyo, Japan

Kazuhisa Seto
Faculty of Information Science and Technology, Hokkaido University, Japan

Takashi Horiyama
Faculty of Information Science and Technology, Hokkaido University, Japan

Abstract

This paper investigates the (quasi-)periodicity of a string when the string is edited. A string C is called a cover (as known as a quasi-period) of a string T if each character of T lies within some occurrence of C. By definition, a cover of T must be a border of T; that is, it occurs both as a prefix and as a suffix of T. In this paper, we focus on the changes in the longest border and the shortest cover of a string when the string is edited only once. We propose a data structure of size $O(n)$ that computes the longest border and the shortest cover of the string in $O(\ell \log n)$ time after an edit operation (either insertion, deletion, or substitution of some string) is applied to the input string T of length n, where ℓ is the length of the string being inserted or substituted. The data structure can be constructed in $O(n)$ time given string T.

2012 ACM Subject Classification Mathematics of computing → Combinatorial algorithms

Keywords and phrases string algorithm, border, cover, quasi-periodicity, dynamic string

Digital Object Identifier 10.4230/LIPIcs.CPM.2024.24

Funding *Takuya Mieno*: JSPS KAKENHI Grant Numbers JP22K21273 and JP23H04381
Kazuhisa Seto: JSPS KAKENHI Grant Number JP22H00513
Takashi Horiyama: JSPS KAKENHI Grant Number JP20H05964

1 Introduction

Periodicity and repetitive structure in strings are important concepts in the field of stringology and have applications in various areas, such as pattern matching and data compression. A string u is called a *period-string* (or simply a *period*) of string T if $T = u^k u'$ holds for some positive integer k and some prefix u' of u. While periods accurately capture the repetitive structure of strings, the definition is too restrictive. In contrast, alternative concepts that capture a sort of periodicity with relaxed conditions have been studied. A *cover* (a.k.a. *quasi-period*) of a string is a typical example of such a concept [5, 6]. A string v is called a cover of T if every character in T lies within some occurrence of v. In other words, T can be written as a repetition of occurrences of v that are allowed to overlap. By definition, a cover of T must occur as both a prefix and a suffix of T, and such string is called a *border* of T. Therefore, a cover of T is necessarily a border of T. For instance, $v =$ `aba` is a cover for $S =$ `abaababa`, and v is both a prefix and a suffix of T. Then, the string $v =$ `aba` of length 3 can be regarded as an "almost" period-string in S while the shortest period-string of S is `abaab` of length 5. Thus, covers can potentially discover quasi-repetitive structures not captured by periods. The concept of covers (initially termed quasi-periods) was introduced by Apostolico and Ehrenfeucht [5, 6]. Subsequently, an algorithm to compute the shortest cover offline in linear time was proposed by Apostolico et al. [7]. Furthermore, an online and linear-time

35th Annual Symposium on Combinatorial Pattern Matching (CPM 2024).
Editors: Shunsuke Inenaga and Simon J. Puglisi; Article No. 24; pp. 24:1–24:15
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

method was presented by Breslauer [9]. Gawrychowski et al. explored cover computations in streaming models [14]. In their problem setting, the computational complexity is stochastic. Other related work on covers can be found in the survey paper by Mhaskar and Smyth [20].

In this paper, we investigate the changes in the shortest cover of a string T when T is edited and design algorithms to compute it. As mentioned above, the shortest cover of T is necessarily a border of T, so we first consider how to compute borders when T is edited. To the best of our knowledge, there is only one explicitly-stated result on the computation of borders in a dynamic setting: the longest border of a string S (equivalently, the smallest period of S) can be maintained in $O(|S|^{o(1)})$ time per character substitution operation (Corollary 19 of [2]). Also, although is not stated explicitly, an $O(\log^3 n)$-time (w.h.p.) algorithm can be obtained by using the results on the *PILLAR model* in dynamic strings [11]. We are unsure whether their results can be applied to compute the shortest cover in a dynamic string. Instead, we focus on studying the changes in covers when a *factor* is edited only once. We believe that this work will be the first step towards the computation of covers for a fully-dynamic string. We now introduce two problems: the LBAE (longest border after-edit) query and the SCAE (shortest cover after-edit) query for the input string T of length n. The LBAE query (resp., the SCAE query) is, given an edit operation *on the original string* T as a query, to compute the longest border (resp., the shortest cover) of the edited string. We note that, after we answer a query, the edit operation is discarded. That is, the following edit operations are also applied to the original string T. This type of problem is called the *after-edit model* [3]. Also, in our problems, the edit operation includes insertion, deletion, or substitution of strings of length one or more. Our main contribution is designing an $O(n)$-size data structure that can answer both LBAE and SCAE queries in $O(\ell \log n)$ time, where ℓ is the length of the string being inserted or substituted. The data structures can be constructed in $O(n)$ time.

Related Work on After-Edit Model

The after-edit model was formulated by Amir et al. [3]. They proposed an algorithm to compute the *longest common factor* (LCF) of two strings in the after-edit model. This problem allows editing operations on only one of the two strings. Abedin et al. [1] subsequently improved their results. Later, Amir et al. [4] generalized this problem to a *fully-dynamic* model and proposed an algorithm that maintains the LCF in $\tilde{O}(n^{\frac{2}{3}})$ time[1] per edit operation. Charalampopoulos et al. [10] improved the maintenance time to amortized $\tilde{O}(1)$ time with high probability per substitution operation. Urabe et al. [23] addressed the problem of computing the *longest Lyndon factor* (LLF) of a string in the after-edit model. The insights gained from their work were later applied to solve the problem of computing the LLF of a fully-dynamic string [4]. Problems of computing the *longest palindromic factor* and *unique palindromic factors* in a string were also considered in the after-edit model [13, 12, 21].

2 Preliminaries

2.1 Basic Definitions and Notations

Strings. Let Σ be an *alphabet*. An element in Σ is called a *character*. An element in $\Sigma^\star$ is called a *string*. The length of a string S is denoted by $|S|$. The string of length 0 is called the *empty string* and is denoted by ε. If a string S can be written as a concatenation of

[1] The $\tilde{O}(\cdot)$ notation hides poly-logarithmic factors.

three strings p, f and s, i.e., $S = pfs$, then p, f and s are called a *prefix*, a *factor*, and a *suffix* of S, respectively. Also, if $|p| < |S|$ holds, p is called a *proper* prefix of S. Similarly, s is called a proper suffix of S if $|s| < |S|$ holds. For any integer i, j with $1 \le i \le j \le |S|$, we denote by $T[i]$ the i-th character of S, and by $T[i..j]$ the factor of S starting at position i and ending at position j. For convenience, let $T[i'..j'] = \varepsilon$ for any i', j' with $i' > j'$. For two strings S and T, we denote by $LCP(S, T)$ the *longest common prefix* of S and T. Also, we denote by $lcp(S, T)$ the length of $LCP(S, T)$. If $f = S[i..i + |f| - 1]$ holds, we say that f *occurs* at position i in S. Let $occ_S(f) = \{i \mid f = S[i..i + |f| - 1]\}$ be the set of occurrences of f in S. Further let $cover_S(f) = \{p \mid p \in [i, i + |f| - 1]$ for some $i \in occ_S(f)\}$ be the set of positions in S that are covered by some occurrence of f in S. A string f is called a *cover* of S if $cover_S(f) = \{1, \dots, |S|\}$ holds. A string b is called a *border* of a non-empty string S if b is both a proper prefix of S and a proper suffix of S. We say that S has a border b when b is a border of S. By definition, any non-empty string has a border ε. If a string S has a border b, integer $p = |S| - |b|$ is called a *period* of S. We sometimes call the smallest period of S *the* period of S. Similarly, we call the longest border of S *the* border of S, and the shortest cover of S *the* cover of S. Also, we denote by $\mathsf{per}(S)$, $\mathsf{bord}(S)$, and $\mathsf{cov}(S)$ the period of S, the border of S, and the cover of S, respectively. The rational number $|S|/\mathsf{per}(S)$ is called the *exponent* of S. We say that S is *periodic* if $\mathsf{per}(S) \le |S|/2$. A string S is said to be *superprimitive* if $\mathsf{cov}(S) = S$.

After-edit Model. The *after-edit model* is, given an edit operation on the input string T as a query, to compute the desired objects on the edited string T' that is obtained by applying the edit operation to T. Note that in the after-edit model, each query, namely each edit operation, is discarded after we finish computing the desired objects on T', so the next edit operation will be applied to the original string T. In this paper, edit operations consist of inserting a string and substituting a factor with another string. Note that factor substitutions contain factor deletions since substituting a factor with the empty string ε is identical to deleting the factor. We denote an edit operation as $\phi(i, j, w)$ where $1 \le j \le |T|$, $1 \le i \le j + 1$ and $w \in \Sigma^\star$: if $i \le j$, $\phi(i, j, w)$ means to substitute $T[i..j]$ for w. If $i = j + 1$, $\phi(i, j, w)$ means to insert w just after $T[i-1]$. In both cases, the resulting string is $T' = T[1..i-1]wT[j+1..|T|]$ and thus $T'[i..i + |w| - 1] = w$. For a given query $\phi(i, j, w)$, let $L_{i,j} = T[1..i - 1]$ and $R_{i,j} = T[j + 1..|T|]$. We will omit the subscripts when they are clear from the context. Thus, $T' = LwR$. We consider the two following problems with the after-edit model:

LBAE (Longest Border After-Edit) query

Preprocess: A string T of length n.
Query: An edit-operation $\phi(i, j, w)$.
Output: The longest border of $T' = L_{i,j}wR_{i,j}$.

SCAE (Shortest Cover After-Edit) query

Preprocess: A string T of length n.
Query: An edit-operation $\phi(i, j, w)$.
Output: The shortest cover of $T' = L_{i,j}wR_{i,j}$.

In the following, we fix the input string T of arbitrary length $n > 0$. Also, we assume that the computation model in this paper is the word-RAM model with word size $\Omega(\log n)$. We further assume that the alphabet Σ is *linearly-sortable*, i.e., we can sort n characters from the input string in $O(n)$ time.

2.2 Combinatorial Properties of Borders and Covers

This subsection describes known properties of borders and covers.

Periodicity of Borders

Let us consider partitioning the set $\mathcal{B}_T$ of borders of a string T of length n. Let $G_1, G_2, \ldots, G_m$ be the sets of borders of T such that $\{G_1, G_2, \ldots, G_m\}$ is a partition of $\mathcal{B}_T$ and for each set, all borders in the same set have the same smallest period. Let p_k be the period of borders belonging to G_k. Without loss of generality, we assume that they are indexed so that $p_k > p_{k+1}$ for every $1 \leq k < m$. We call G_k the *k-th group*. Then, the following fact is known:

▶ **Proposition 1** ([17, 16]). *For the partition $\{G_1, G_2, \ldots, G_m\}$ of $\mathcal{B}_T$ defined above, the following statements hold:*
1. *For each $1 \leq k \leq m$, the lengths of borders in the k-th group can be represented as a single arithmetic progression with common difference p_k.*
2. *If a group contains at least three elements, the borders in the group except for the shortest one are guaranteed to be periodic.*
3. *The number m of sets is in $O(\log n)$.*

Properties of Covers

The following lemma summarizes some basic properties of covers, which we will use later.

▶ **Lemma 2** ([9, 22]). *For any string T, the following statements hold.*
1. *The cover $\mathsf{cov}(T)$ of T is either $\mathsf{cov}(\mathsf{bord}(T))$ or T itself.*
2. *$\mathsf{cov}(T)$ is superprimitive and non-periodic.*
3. *Let v be a cover of T, and u be a factor of T which is shorter than v. Then u is a cover of T if and only if u is a cover of v.*

2.3 Algorithmic Tools

This subsection shows algorithmic tools we will use later.

Border Array and Border-group Array

The *border array* B_T of a string T is an array of length n such that $\mathsf{B}_T[i]$ stores the length of the border of $T[1..i]$ for each $1 \leq i \leq n$. Also, for convenience, let $\mathsf{B}_T[0] = 0$ for any string T. There is a well-known *online* algorithm for linear-time computation of the border array (e.g., see [15]). While the worst-case running time of the algorithm is $O(n)$ per a character, it can be made $O(\log n)$ by constructing B with the *strict border array* proposed in [17].

▶ **Lemma 3** ([15, 17]). *For each $1 \leq i \leq n$, if we have $T[1..i-1]$ and $\mathsf{B}_{T[1..i-1]}$, then we can compute $\mathsf{B}_{T[1..i]}$ in worst-case $O(\log n)$ time and amortized $O(1)$ time given the next character $T[i]$.*

Next, we introduce a data structure closely related to the border array. The *border-group array* BG_T of a string T is an array of length n such that, for each $1 \leq i \leq n$, $\mathsf{BG}_T[i]$ stores the length of the shortest border of $T[1..i]$ whose smallest period equals $\mathsf{per}(T[1..i])$ if such a border exists, and $\mathsf{BG}_T[i] = i$ otherwise. By definition, if $T[1..i]$ is a border of T and it belongs to group G_k, then $\mathsf{BG}_T[i]$ stores the first (the smallest) term of the arithmetic

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
$T[i]$		a	b	a	b	a	b	a	a	b	a	b	a	b	a	a	b	a
$\mathsf{B}_T[i]$	0	0	0	1	2	3	4	5	1	2	3	4	5	6	7	8	9	10
$\mathsf{per}(T[1..\,i])$		1	2	2	2	2	2	2	7	7	7	7	7	7	7	7	7	7
$\mathsf{per}(\mathsf{bord}(T[1..\,i]))$		-	-	1	2	2	2	2	1	2	2	2	2	2	2	7	7	7
$\mathsf{BG}_T[i]$		1	2	3	2	3	2	3	8	9	10	11	12	13	14	8	9	10

Figure 1 An example of a border array and a border-group array. For position $i = 7$, the period of $T[1..7]$ is 2. All borders of $T[1..7]$ are ababa, aba, a, and ε. Also, their smallest periods are 2, 2, 1, and 0, respectively. Thus $\mathsf{BG}_T[7] = |\mathtt{aba}| = 3$. For position $i = 8$, the period of $T[1..8]$ is 7. Any border of $T[1..8]$ does not have period 7, and thus $\mathsf{BG}_T[8] = i = 8$.

progression representing the lengths of borders in G_k. This is why we named BG_T the border-group array. Also, the common difference $p_k = i - \mathsf{B}_T[i]$ can be obtained from B_T if $\mathsf{BG}_T[i] \neq i$. See Figure 1 for an example. We can compute the border-group array in linear time in an online manner together with B_T. Before proving it, we note a fact about periods.

▶ **Proposition 4.** *If u is a factor of v, then $\mathsf{per}(u) \leq \mathsf{per}(v)$.*

▶ **Lemma 5.** *For each $1 < i \leq n$, if we have $T[1..i-1]$, $\mathsf{B}_{T[1..i-1]}$, and $\mathsf{BG}_{T[1..i-1]}$, then we can compute $\mathsf{BG}_{T[1..i]}$ in amortized $O(1)$ time given the next character $T[i]$.*

Proof. By definition, $\mathsf{B}_{T[1..1]} = [0]$ and $\mathsf{BG}_{T[1..1]} = [1]$. Assume that we have $\mathsf{B}_{T[1..i-1]}$, $\mathsf{BG}_{T[1..i-1]}$, and $T[1..i]$ for $i \geq 2$. By Lemma 3, we can obtain $\mathsf{B}_{T[1..i]}$ in amortized $O(1)$ time. Now let $p = i - \mathsf{B}_{T[1..i]}[i]$ and $q = \mathsf{B}_{T[1..i]}[i] - \mathsf{B}_{T[1..i]}[\mathsf{B}_{T[1..i]}[i]]$, meaning that $p = \mathsf{per}(T[1..i])$ and $q = \mathsf{per}(\mathsf{bord}(T[1..i]))$. If $p = q$, then we set $\mathsf{BG}_{T[1..i]}[i] = \mathsf{BG}_{T[1..i-1]}[\mathsf{B}_{T[1..i]}[i]]$ since $\mathsf{per}(T[1..i]) = \mathsf{per}(\mathsf{bord}(T[1..i]))$. Otherwise, $\mathsf{per}(T[1..i]) > \mathsf{per}(\mathsf{bord}(T[1..i]))$, and thus, the period of *any* border of $T[1..i]$ is smaller than $\mathsf{per}(T[1..i])$ by Proposition 4. Hence we set $\mathsf{BG}_{T[1..i]}[i] = i$. The running time of the algorithm is (amortized) $O(1)$. ◀

Longest Common Extension Query

The *longest common extension* query (in short, *LCE* query) is, given positions i and j within T, to compute $lcp(T[i..|T|], T[j..|T|])$. We denote the answer of the query as $lce_T(i, j)$. We heavily use the following result in our algorithms.

▶ **Lemma 6** (E.g., [8]). *We can answer any LCE query in $O(1)$ time after $O(n)$-time and space preprocessing on the input string T.*

Prefix Table

The *prefix table* Z_S of a string S of length m is an array of length m such that $\mathsf{Z}_S[i] = LCP(S, S[i..m])$ for each $1 \leq i \leq m$.

▶ **Lemma 7** ([19, 15]). *Given a string S of length m over a general unordered alphabet, we can compute the prefix table Z_S in $O(m)$ time*[2].

We emphasize that this linear-time algorithm does not require linearly-sortability of the alphabet.

[2] The algorithm described in [15] is known as *Z-algorithm*, so we use Z to represent the prefix table.

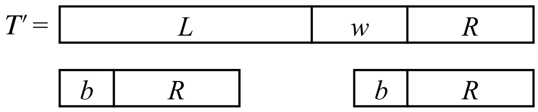

Figure 2 A border of T' which is longer than R is written as bR where b is a border of Lw.

Internal Pattern Matching

The *internal pattern matching* query (in short, *IPM* query) is, given two factors u, v of T with $|v| \leq 2|u|$, to compute the occurrences of u in v. The output is represented as an arithmetic progression due to the lengths constraint and periodicity [18]. If u occurs in v, we denote by $\mathsf{rightend}(u, v)$ the ending position of the rightmost occurrence of u in v.

▶ **Lemma 8** ([18]). *We can answer any IPM query in $O(1)$ time after $O(n)$-time and space preprocessing on the input string T,*

3 Longest Border After Edit

This section proposes an algorithm to solve the LBAE problem. In the following, we assume that $|L| \geq |R|$ and $|w| \leq |L|/2$ for a fixed query $\phi(i, j, w)$. Because, when $|L| < |R|$, running our algorithm on the reversal inputs can answer LBAE queries without growing complexities. Also, if $|w| > |L|/2$, then $|w| > |T'|/5$ holds since $T' = LwR$ and $|L| \geq |R|$. We can obtain the border of T' in $O(|T'|) = O(|w|)$ time by computing the border array of T' from scratch.

We compute the border of T' in the following two steps. **Step 1**: Find the longest border of T' which is longer than R. **Step 2**: Find the longest border of T' of length at most $|R|$ if nothing is found in step 1. Step 2 can be done in constant time by pre-computing all borders of T and the longest border of T of length at most k for each k with $1 \leq k \leq n$. Thus we focus on step 1, i.e., how to find the longest border of T' which is longer than R. We observe that such a border is the concatenation of some border of Lw and R (see Figure 2). By pre-computing the border array B_T, the border array B_{Lw} of Lw can be computed in $O(|w| \log n)$ time starting from $\mathsf{B}_L = \mathsf{B}_T[1..|L|]$ (Lemma 3). Let b_{Lw} be the border of Lw. There are two cases: (i) $|b_{Lw}| \leq |w|$ or (ii) $|b_{Lw}| > |w|$. We call the former case the short border case and the latter case the long border case.

3.1 Short Border Case

In this case, the length of the border of $T' = LwR$ is at most $|b_{Lw}R| \leq |wR| \leq |Lw|$, so its prefix-occurrence ends within Lw. Also, for any border b of Lw, string bR is a border of T' if and only if $lce_{T'}(|b| + 1, |Lw| + 1) = |R|$ holds. Thus, we pick up each border b of Lw in descending order of length and check whether bR is a border of T' by computing $lce_{T'}(|b| + 1, |Lw| + 1)$. Since Lw has at most $|w|$ borders, constant-time LCE computation results in a total of $O(|w|)$ time. If $|b| + |R| \leq |L|$ then we can use the LCE data structure on the original string T since $lce_{T'}(|b| + 1, |Lw| + 1) = lce_T(|b| + 1, j + 1)$ holds. Otherwise, we may compute the longest common prefix of w and some suffix of R, which cannot be computed by applying LCE queries on T naïvely. To resolve this issue, we compute the prefix table Z_W of W in $O(|W|) = O(|w|)$ time where $W = w \cdot R[|R| - |w| + 1..|R|]$ is the concatenation of w and the length-$|w|$ suffix of R. Note that $|R| > |w|$ holds here since $|R| > |L| - |b| \geq |L| - |w| \geq |w|$ by the assumptions in this case. Then the longest common prefix of w and any suffix of R of length at most $|w|$ is obtained in constant time, and so is $lce_{T'}(|b| + 1, |Lw| + 1)$. Therefore, we can compute $lce_{T'}(|b| + 1, |Lw| + 1)$ for all borders b of Lw in a total of $O(|w| \log n)$ time.

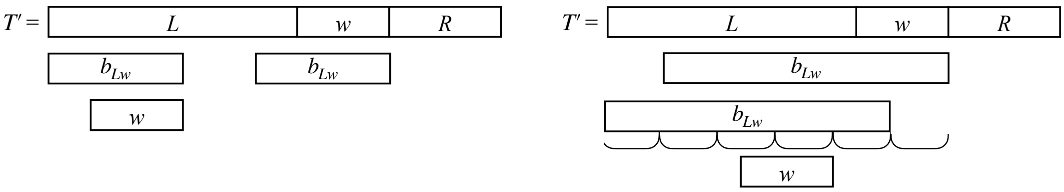

Figure 3 Left: If b_{Lw} is not longer than L, then w occurs within L as a suffix of the prefix-occurrence of b_{Lw} in L. Right: If b_{Lw} is longer than L, then w occurs within L because of the periodicity of b_{Lw}.

3.2 Long Border Case

Firstly, we give some observations for the long border case; $|b_{Lw}| > |w|$. If $|b_{Lw}| \leq |L|$, then $w = L[|b_{Lw}| - |w| + 1..|b_{Lw}|]$ holds. If $|b_{Lw}| > |L|$, then the period p_{Lw} of Lw is $p_{Lw} = |Lw| - |b_{Lw}| < |Lw| - |L| = |w|$. Let k be the smallest integer such that $kp_{Lw} \geq |w|$. Since $k \geq 2$, $kp_{Lw} \leq 2(k-1)p_{Lw} < 2|w| \leq |L|$ holds. Thus $w = L[|L| - kp_{Lw} + 1..|L| - kp_{Lw} + |w|]$ holds (see also Figure 3). Thus, in both cases, w occurs *within* L, which is a factor of the original T. From this observation, any single LCE query on T' can be simulated by constant times LCE queries on T because any LCE query on $w = T'[|L|+1..|L|+|w|]$ can be simulated by a constant number of LCE queries on another occurrence of w within L. Therefore, in the following, we use the fact that any LCE query on $T' = LwR$ can be answered in $O(1)$ time as a black box.

Now, we show some properties of the border of T'. As we mentioned in Proposition 1, the sets of borders of Lw can be partitioned into $m \in O(\log n)$ groups w.r.t. their smallest periods. Let $G_1, G_2, \ldots, G_m$ be the groups such that $p_k > p_{k+1}$ for every $1 \leq k < m$, where p_k is the period of borders in G_k. Next, let us assume that there exists a border of T' which is longer than R. Let $b^\star$ be the border of Lw such that $b^\star R$ is the border of T'. Further let $k^\star$ be the index of the group to which $b^\star$ belongs. There are three cases: (i) $b^\star$ is periodic and $\mathsf{per}(b^\star) = p_{k^\star} = \mathsf{per}(b^\star R)$, (ii) $b^\star$ is periodic and $\mathsf{per}(b^\star) = p_{k^\star} \neq \mathsf{per}(b^\star R)$, or (iii) $b^\star$ is not periodic. The first two cases are illustrated in Figure 4. For the case (i), the following lemma holds. Here, for a group G_k, let α_k be the exponent of the longest prefix of T' with period p_k.

▶ **Lemma 9.** *If $b^\star$ is periodic and $p_{k^\star} = \mathsf{per}(b^\star R)$, then $|b^\star| \leq \alpha_{k^\star} p_{k^\star} - |R|$ holds.*

Proof. Assume the contrary that $|b^\star| > \alpha_{k^\star} p_{k^\star} - |R|$ holds. Then $|b^\star R|/p_{k^\star} > \alpha_{k^\star}$ holds. This contradicts the maximality of $\alpha_{k^\star}$ since $b^\star R$ occurs as a prefix of T' and $\mathsf{per}(b^\star R) = p_{k^\star}$. ◀

For the case (ii), the following lemma holds. Here, for a group G_k, let $r_k = lce_{T'}(|T'| - |R| - p_k + 1, |T'| - |R| + 1)$.

▶ **Lemma 10.** *If $b^\star$ is periodic and $p_{k^\star} \neq \mathsf{per}(b^\star R)$, then $|b^\star| = \alpha_{k^\star} p_{k^\star} - r_{k^\star}$ holds.*

Proof. Since the period of $b^\star$ is $p_{k^\star}$, the longest prefix of $T'[|T'| - |b^\star R| + 1..|T'|]$ with period $p_{k^\star}$ is of length $|b^\star| + r_{k^\star}$. Thus, by the definition of $\alpha_{k^\star}$, $\alpha_{k^\star} p_{k^\star} = |b^\star| + r_{k^\star}$ holds since $T'[1..|b^\star R|] = T'[|T'| - |b^\star R| + 1..|T'|]$. Therefore, $|b^\star| = \alpha_{k^\star} p_{k^\star} - r_{k^\star}$. ◀

Clearly, if $p_{k^\star} = \mathsf{per}(b^\star R)$, then $r_{k^\star} = |R|$ holds. Hence, by combining the two above lemmas, we obtain the next corollary:

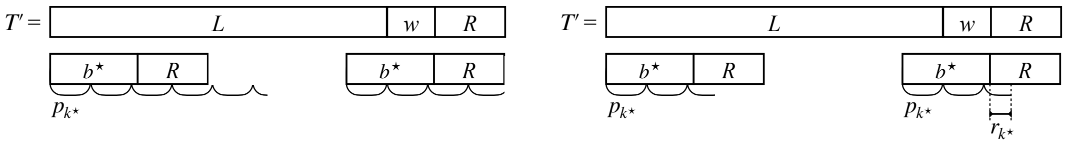

Figure 4 Left: Illustration for the case (i) $b^\star$ is periodic and $\mathsf{per}(b^\star) = p_{k^\star} = \mathsf{per}(b^\star R)$. The period $p_{k^\star}$ repeats five times and a little more in T'. Then $|b^\star|$ is at most the length of the repetition minus $|R|$ (Lemma 9). Right: Illustration for the case (ii) $b^\star$ is periodic and $\mathsf{per}(b^\star) = p_{k^\star} \neq \mathsf{per}(b^\star R)$. Since the maximal repetition of period $p_{k^\star}$ ends within R, the length $|b|$ is equal to the length of the maximal repetition minus $r_{k^\star}$ where $r_{k^\star}$ is the length of the suffix of the repetition that enters R (Lemma 10).

▶ **Corollary 11.** *If $b^\star$ is periodic, then $b^\star$ is the longest border of Lw whose length is at most $\alpha_{k^\star} p_{k^\star} - r_{k^\star}$.*

Based on this corollary, we design an algorithm to answer the LBAE queries.

Algorithm

The idea of our algorithm is as follows: given a query, we first initialize *candidates-set* $\mathcal{C} = \emptyset$, which will be a set of candidates for the length of the border of T'. Next, for each group of borders of Lw, we calculate a constant number of candidates from the group and add their lengths to the candidates-set $\mathcal{C}$ (the details are described below). In the end, we choose the maximum from $\mathcal{C}$ and output it.

Now we consider the k-th group G_k for a fixed k and how to calculate candidates. If $|G_k| \leq 2$, we just try to extend each border in G_k to the right by using LCE queries on T', and if the extension reaches the right-end of T', we add its length to $\mathcal{C}$. Note that we do not care about the periodicity of borders here. Otherwise, we compute α_k and r_k by using LCE queries on T'. Let $\tilde{b}_k$ be the longest element in G_k whose length is at most $\alpha_k p_k - r_k$, if such a border exists. If $\tilde{b}_k$ is defined, we check whether $\tilde{b}_k R$ is a border of T' or not, again by using an LCE query on T'. If $\tilde{b}_k R$ is a border of T', we add its length to $\mathcal{C}$. Also, we similarly check whether $b_k^{\min} R$ is a border of T', and if so, add its length to $\mathcal{C}$, where $b_k^{\min}$ is the shortest element in G_k, which may be non-periodic.

Correctness

If a group G_k contains at least three elements, the borders in G_k except for the shortest one are periodic (Proposition 1). Namely, any non-periodic border of Lw is either an element of a group whose size is at most two, or the shortest element of a group whose size is at least three. Both cases are completely taken care of by our algorithm. For periodic borders, it is sufficient to check the longest border $\tilde{b}_k$ of Lw whose length is at most $\alpha_k p_k - r_k$ for each group G_k by Corollary 11. Therefore, the length of the border of T' must belong to the candidates-set $\mathcal{C}$ obtained at the end of our algorithm.

Running Time

Given a query $\phi(i, j, w)$, we can obtain the border array B_{Lw} and the border-group array BG_{Lw} in $O(|w| \log n)$ time from $\mathsf{B}_T[1..|L|]$ and $\mathsf{BG}_T[1..|L|]$ (Lemmas 3 and 5). Thus, by using those arrays, while we scan the groups $G_1, \ldots, G_m$, we can determine whether the current group G_k has at least three elements or not, and compute the first term and the common difference of the arithmetic progression representing the current group G_k both in constant time. All the other operations consist of LCE queries on T' and basic arithmetic operations, which can be done in constant time. Finally, we choose the maximum from $\mathcal{C}$, which can be done $O(|\mathcal{C}|)$ time. Since we add at most two elements to $\mathcal{C}$ when we process each group, the size of $\mathcal{C}$ is in $O(m)$. Thus the total running time is in $O(|w| \log n + m) \subseteq O(|w| \log n)$ since $m \in O(\log n)$.

To summarize this section, we obtain the following theorem.

▶ **Theorem 12.** *The longest border after-edit query can be answered in $O(\ell \log n)$ time after $O(n)$-time preprocessing, where ℓ is the length of the string to be inserted or substituted specified in the query.*

4 Shortest Cover After Edit

This section proposes an algorithm to solve the SCAE problem. Firstly, we give additional notations and tools. For a string S and an integer k with $1 \leq k \leq |S|$, $\mathsf{range}(S, k)$ denotes the largest integer r such that $S[1..k]$ can cover $S[1..r]$. Next, we give definitions of two arrays $\mathsf{C}(T)$ and $\mathsf{R}(T)$ introduced in [9]. The former $\mathsf{C}(T)$ is called the *cover array* and stores the length of the cover of each prefix of T, i.e., $\mathsf{C}(T)[k] = |\mathsf{cov}(T[1..k])|$ for each k with $1 \leq k \leq n$. For convenience, let $\mathsf{C}(T)[0] = 0$. The latter $\mathsf{R}(T)$ is called the *range array* that stores the values of range function only for *superprimitive prefixes* of T, i.e., for each k, $\mathsf{R}(T)[k] = \mathsf{range}(T, k)$ if $\mathsf{cov}(T[1..k]) = T[1..k]$, and otherwise $\mathsf{R}(T)[k] = 0$, meaning undefined. Cover array and range array can be computed in $O(n)$ time given T in an online manner [9]. In describing our algorithm, we use the next lemma:

▶ **Lemma 13.** *Assume that we already have data structure $\mathcal{D}_T$ consisting of the IPM data structure on T of Lemma 8, border array $\mathsf{B}(T)$, cover array $\mathsf{C}(T)$, range array $\mathsf{R}(T)$, and an array $\mathsf{R}^\star$ of size n initialized with $\mathbf{0}$. Given a query $\phi(i, j, w)$, we can enhance $\mathcal{D}_T$ in $O(|w| \log n)$ time so that we can obtain $\mathsf{cov}((Lw)[1..k])$ for any k with $1 \leq k \leq |Lw|$ and $\mathsf{range}(Lw, k')$ for any k' such that $1 \leq k' \leq |L|$ and $\mathsf{cov}(L[1..k']) = L[1..k']$ in $O(1)$ time.*

Due to lack of space, we only give the idea of the proof of Lemma 13. To prove the lemma, we first review Breslauer's algorithm [9] that computes $\mathsf{C}(T)$ and $\mathsf{R}(T)$ for a given string T in an online manner (Algorithm 1). By Lemma 3, we can compute $\mathsf{B}(Lw)$ in $O(|w| \log n)$ time if $\mathsf{B}(L)$ and w are given. Since Algorithm 1 runs in an online manner, if we have $\mathsf{C}(L)$ and $\mathsf{R}(L)$ in addition to $\mathsf{B}(Lw)$, then it is easy to obtain $\mathsf{C}(Lw)$ and $\mathsf{R}(Lw)$ in $O(|w|)$ time by running Algorithm 1 starting from the $(|L| + 1)$-th iteration. However, we only have $\mathsf{C}(T)$ and $\mathsf{R}(T)$, not $\mathsf{C}(L)$ and $\mathsf{R}(L)$. The idea of our algorithm for Lemma 13 is to simulate $\mathsf{C}(L \cdot w[1..t-1])$ and $\mathsf{R}(L \cdot w[1..t-1])$ while iterating the while-loop of Algorithm 1 from $t = 1$ ($idx = |L| + 1$) to $t = |w|$ ($idx = |L| + |w|$). We also show a complete pseudocode in Algorithm 2.

Note that we can prepare the input $\mathcal{D}_T$ of Lemma 13 in $O(n)$ time for a given T.

Overview of Our Algorithm for SCAE Queries

To compute the cover of T', we first run the LBAE algorithm of Section 3. Then, there are two cases: (i) The *non-periodic case*, where the length of $\mathsf{bord}(T')$ is smaller than $|T'|/2$, or (ii) the *periodic case*, the other case.

Algorithm 1 Algorithm to compute $\mathsf{C}(T)$ proposed in [9].

Require: The border array $\mathsf{B}(T)$ of string T, and two arrays $C[0..n] = R[0..n] = \mathbf{0}$.
Ensure: $C = \mathsf{C}(T)$ and $R = \mathsf{R}(T)$

$idx \leftarrow 1$
while $idx \leq n$ **do**
  $clen \leftarrow C[\mathsf{B}(T)[idx]]$ ▷ $clen < idx$ always holds.
  if $clen > 0$ and $R[clen] \geq idx - clen$ **then**
    $C[idx] \leftarrow clen$
    $R[clen] \leftarrow idx$ ▷ When $T[1..idx]$ is *not* superprimitive, $R[clen]$ is updated to idx.
  else
    $C[idx] \leftarrow idx$
    $R[idx] \leftarrow idx$ ▷ When $T[1..idx]$ is superprimitive, $R[idx]$ is newly defined.
  end if
  $idx \leftarrow idx + 1$
end while

4.1 Non-periodic Case

Let $b = \mathsf{bord}(T')$ and $c = \mathsf{cov}(b)$. By the first statement of Lemma 2, $\mathsf{cov}(T') = \mathsf{cov}(\mathsf{bord}(T'))$ if $\mathsf{cov}(\mathsf{bord}(T'))$ can cover T', and $\mathsf{cov}(T') = T'$ otherwise. In the following, we consider how to determine whether $c = \mathsf{cov}(\mathsf{bord}(T'))$ is a cover of T' or not.

Let s be the maximum length of the prefix of Lw that c can cover. Further let t be the maximum length of the suffix of wR that c can cover if $|c| \leq |wR|$, and $t = |c|$ otherwise. By Lemma 13, the values of s and t can be obtained in $O(|w| \log n)$ time by computing values of $\mathsf{range}(Lw, |c|)$ and $\mathsf{range}((wR)^R, |c|)$ since $c = \mathsf{cov}(b)$ is superprimitive (by the second statement of Lemma 2), where $(wR)^R$ denotes the reversal of wR. If $s + t \geq |T'|$ then c is a cover of T'. Thus, $\mathsf{cov}(T') = c$, and the algorithm is terminated.

We consider the other case, where $s + t < |T'|$. The inequality $s + t < |T'|$ means that the occurrences of c within Lw or wR cannot cover the middle factor $T'[s + 1..|T'| - t]$ of T'. Thus, if c is a cover of T' when $s + t < |T'|$, then c must have an occurrence that starts in L and ends in R. Such an occurrence can be written as a concatenation of some border of Lw which is longer than w and some (non-empty) prefix of R. Similar to the method in Section 3.2, we group the borders of Lw using their periods and process them for each group. Again, let $G_1, \ldots, G_m$ be the groups sorted in descending order of their smallest periods.

Let us fix a group G_k arbitrarily. If $|G_k| \leq 2$, we simply try to extend each border in G_k to the right by using LCE queries. Now we use the following claim:

▷ Claim 14. For a border z of Lw with $|z| > |w|$, the value of $lce_{T'}(|z| + 1, |Lw| + 1)$ can be computed in constant time by using the LCE data structure of Lemma 6 on T.

This claim can be proven by similar arguments as in the first paragraph of Section 3.2. Thus, the case of $|G_k| \leq 2$ can be processed in constant time. If $|G_k| > 2$, we use the period p_k of borders in G_k. Let α_k be the exponent of the longest prefix of T' with period p_k. Further let $r_k = lce_{T'}(|T'| - |R| - p_k + 1, |T'| - |R| + 1)$. Note that $\alpha_k p_k < |c|$ since c is a prefix of T' and is non-periodic. See also Figure 5. By using LCE queries on T', α_k and r_k can be computed in constant time. For a border z in G_k, $T'[|z| + 1..|c|] = T'[|Lw| + 1..|Lw| + |c| - |z|]$ holds only if $|z| = \alpha_k p_k - r_k$. Thus, the only candidate for a border in G_k which can be extended to the right enough is of length exactly $\alpha_k p_k - r_k$ if it exists. The existence of such a border can be determined in constant time since the lengths of the borders in G_k are represented

Algorithm 2 Algorithm to compute data structures which can simulate $\mathsf{C}(Lw)$ and $\mathsf{R}(Lw)$.

Require: $\mathsf{B}(T)$, $\mathsf{C}(T)$, $\mathsf{R}(T)$, $\mathsf{R}^\star[1..n] = \mathbf{0}$, and $\phi(i, j, w)$.
Ensure: (i) $C_w[1..|w|] = \mathsf{C}(Lw)[|L|+1..|Lw|]$ and (ii) $\mathsf{R}^\star[k] = \mathsf{R}(Lw)[k]$ if $1 \le k \le |Lw|$ and $\mathsf{R}(T)[k] \neq \mathsf{R}(Lw)[k] > |L|$, and $\mathsf{R}^\star[k] = 0$ otherwise.

$idx \leftarrow |L| + 1$ ▷ Starting from $(|L|+1)$-th position.
while $idx \le |Lw|$ **do**
  $clen \leftarrow \mathsf{C}(T)[\mathsf{B}(T)[idx]]$
  if $clen > 0$ **then** ▷ $T[1..clen]$ is superprimitive.
    if $\mathsf{R}^\star[clen] \neq 0$ **then**
      $r \leftarrow \mathsf{R}^\star[clen]$
    else if $\mathsf{R}(T)[clen] \le |L|$ **then**
      $r \leftarrow \mathsf{R}(T)[clen]$
    else ▷ $\mathsf{R}^\star[clen] = 0$ and $\mathsf{R}(T)[clen] > |L|$
      $r \leftarrow \mathsf{rightend}(T[1..clen], T[|L| - 2clen + 2..|L|])$
    end if
    if $r \ge idx - clen$ **then** ▷ $r = \mathsf{R}(L \cdot w[1..idx - |L|])[clen]$
      $C_w[idx - |L|] \leftarrow clen$
      $\mathsf{R}^\star[clen] \leftarrow idx$ ▷ $(Lw)[1..idx]$ is *not* superprimitive.
      $idx \leftarrow idx + 1$
      continue ▷ Go to the next iteration.
    end if
  end if
  $C_w[idx - |L|] \leftarrow idx$
  $\mathsf{R}^\star[idx] \leftarrow idx$ ▷ $(Lw)[1..idx]$ is superprimitive.
  $idx \leftarrow idx + 1$
end while

as an arithmetic progression. If such a border of length $\alpha_k p_k - r_k$ exists, then we check whether it can be extended to the desired string c by querying an LCE. Therefore, the total computation time is $O(1)$ for a single group G_k, and $O(\log n)$ time in total for all groups since there are $O(\log n)$ groups.

To summarize, we can compute $\mathsf{cov}(T')$ in $O(|w| \log n)$ for the non-periodic case.

4.2 Periodic Case

In this case, T' can be written as $(uv)^k u$ for some integer $k \ge 2$ and strings u, v with $|uv| = \mathsf{per}(T')$ since T' is periodic. By the third statement of Lemma 2, $\mathsf{cov}(T') = \mathsf{cov}(uvu)$ holds since uvu is a cover of T'. Thus, in the following, we focus on how to compute $\mathsf{cov}(uvu)$. We further divide this case into two sub-cases depending on the relation between the lengths of uvu and Lw.

If $|uvu| \le |Lw|$, then uvu is a prefix of Lw. Thus, by Lemma 13, $\mathsf{cov}(uvu) = \mathsf{cov}((Lw)[1..|uvu|])$ can be computed in $O(|w| \log n)$ time.

If $|uvu| > |Lw|$, then $T' = uvuvu$ since $|uvu| > n/2$. We call the factor $T[|uv|+1..|uvu|] = u$ the *second occurrence of* u. Also, since $|L| \ge |R| = |T'| - |Lw| > |T'| - |uvu| = |vu|$, both R and L are longer than uv. Thus vu is a suffix of R and uv is a prefix of L. Now let us consider the border of uvu.

▶ **Lemma 15.** *If the period of a string $T' = uvuvu$ is $|uv|$, then the border of uvu is not longer than $|uv|$.*

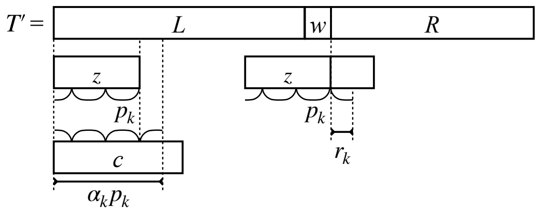

Figure 5 Illustration for the non-periodic case. Here, c is non-periodic, z is some border of Lw, and p_k is the period of z. If there is an occurrence of c starting in R and ending in R, then $|z| = \alpha_k p_k - r_k$ must hold since c is non-periodic.

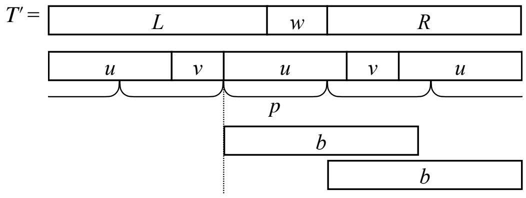

Figure 6 Illustration for a contradiction if we assume that uvu has a border which is longer than $|uv|$. Since $T' = uvuvu$, if uvu has a period which is smaller than $|u|$ then T' also has the same period.

Proof. If uvu has a border that is longer than $|uv|$, uvu has a period p which is smaller than $|u|$. Then the length-p prefix of the second occurrence of u repeats to the left and the right until it reaches both ends of T' (see Figure 6). This contradicts that $\mathsf{per}(T') = |uv|$. ◀

Therefore, the border of uvu is identical to the longest border of T whose length is at most $|uv|$, which can be obtained in constant time after $O(n)$-time preprocessing as in step 2 of Section 3. By the first statement of Lemma 2, $\mathsf{cov}(uvu)$ is either $\mathsf{cov}(\mathsf{bord}(uvu))$ or uvu. Since $|\mathsf{bord}(uvu)| \leq |uv| < |Lw|$, $\mathsf{cov}(\mathsf{bord}(uvu))$ can be obtained in $O(|w| \log n)$ time by Lemma 13. Let $x = \mathsf{cov}(\mathsf{bord}(uvu))$. Thanks to Lemma 16 below, we do not have to scan $O(\log n)$ groups, unlike the non-periodic case.

▶ **Lemma 16.** *When $|uvu| > |Lw|$, string $x = \mathsf{cov}(\mathsf{bord}(uvu))$ covers uvu if and only if $\mathsf{range}(Lw, |x|) \geq |uvu| - \max\{|u|, |x|\}$ holds.*

Proof. Let $r = \mathsf{range}(Lw, |x|)$. We divide the proof into three cases.

The case when $|x| \leq |u|/2$: In this case, x is a border of u and the occurrence of x as the prefix of the second occurrence of u ends within Lw. ($\Longrightarrow$) If x covers uvu, then $r \geq |uv| + |x| > |uv| = |uvu| - |u|$. ($\Longleftarrow$) If $r \geq |uvu| - |u| = |uv|$ holds, then x covers uvx and u. Hence x covers uvu (see the left figure of Figure 7).

The case when $|u|/2 < |x| \leq |u|$: In this case, x is a border of u and its prefix-suffix occurrences in u share the center position $\lceil |u|/2 \rceil$ of u. ($\Longrightarrow$) Assume the contrary, i.e., x covers uvu and $r < |uv|$. Since x covers uvu, there exists an occurrence of x that covers position $r + 1$. Also, since $r < |uv|$, the occurrence does not end within Lw. Thus, the occurrence must cover the center position $\lceil |u|/2 \rceil$ of the second occurrence of u. Now, there are three distinct occurrences of x that cover the same position $\lceil |u|/2 \rceil$, however, it contradicts that $x = \mathsf{cov}(\mathsf{bord}(uvu))$ is non-periodic (the second statement of Lemma 2). ($\Longleftarrow$) Similar to the previous case, if $r \geq |uvu| - |u| = |uv|$ holds, then x covers uvx and u. Hence x covers uvu.

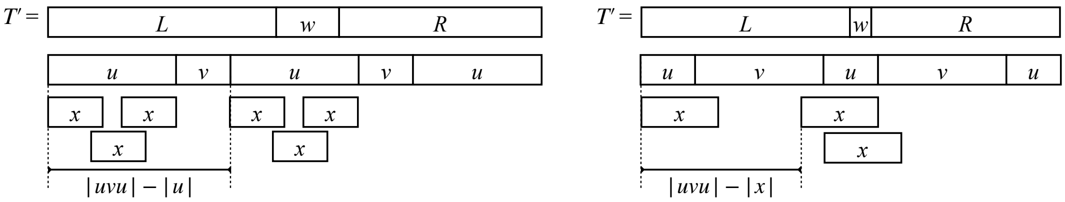

Figure 7 Left: Illustration for the case $|x| \leq |u|$. Right: Illustration for the case $|x| > |u|$.

The case when $|x| > |u|$: Let $s = |uvu| - |x|$. In this case, x occurs at positions $s+1$ and $|uv|+1$. Thus, the occurrences share position $|uv|+1$, which is the first position of the second occurrence of u (see the right figure of Figure 7). ($\Longrightarrow$) Assume the contrary, i.e., x covers uvu and $r < s$. Similar to the previous case, there must be an occurrence of x such that the occurrence covers position $r+1$ and does not end within Lw. Then, there are three distinct occurrences of x that cover the same position $|uv|+1$, which leads to a contradiction with the fact that x is non-periodic. ($\Longleftarrow$) This statement is trivial by the definitions and the conditions. ◀

Therefore, if $\mathsf{range}(Lw, |x|) \geq |uvu| - \max\{|u|, |x|\}$ then the cover of uvu is x. Otherwise, the cover of uvu is uvu itself. Further, by Lemma 13, the value of $\mathsf{range}(Lw, |x|)$ can be obtained in $O(|w| \log n)$ time since $x = \mathsf{cov}(\mathsf{bord}(uvu))$ is superprimitive and $|x| \leq |uv| < |Lw|$.

To summarize, we can compute $\mathsf{cov}(T')$ in $O(|w| \log n)$ for the periodic case.

Finally, we have shown the main theorem of this paper:

▶ **Theorem 17.** *The shortest cover after-edit query can be answered in $O(\ell \log n)$ time after $O(n)$-time preprocessing, where ℓ is the length of the string to be inserted or substituted specified in the query.*

5 Conclusions and Discussions

In this paper, we introduced the problem of computing the longest border and the shortest cover in the after-edit model. For each problem, we proposed a data structure that can be constructed in $O(n)$ time and can answer any query in $O(\ell \log n)$ time where n is the length of the input string, and ℓ is the length of the string to be inserted or replaced.

As a direction for future research, we are interested in improving the running time. For LBAE queries, when the edit operation involves a single character, an $O(\log(\min\{\log n, \sigma\}))$ query time can be achieved by exploiting the periodicity of the border: we pre-compute all *one-mismatch borders* and store the triple of mismatch position, mismatch character, and the mismatch border length for each mismatch border. The number of such triples is in $O(n)$. Furthermore, the number of triples for each position is $O(\min\{\log n, \sigma\})$ due to the periodicity of borders. Thus, by employing a binary search on the triples for the query position, the query time is $O(\log(\min\{\log n, \sigma\}))$. However, this algorithm stores all mismatch borders and cannot be straightforwardly extended to editing strings of length two or more. It is an open question whether the query time of LBAE and SCAE queries can be improved to $O(\ell + \log \log n)$ for an edit operation of length-ℓ string in general. Furthermore, applying the results obtained in this paper to a more general problem setting, particularly the computation of borders/covers in a fully-dynamic string, is a future work that needs further exploration.

References

1 Paniz Abedin, Sahar Hooshmand, Arnab Ganguly, and Sharma V. Thankachan. The heaviest induced ancestors problem revisited. In *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, Itai Boneh, Panagiotis Charalampopoulos, and Eitan Kondratovsky. Repetition detection in a dynamic string. In *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.

3 Amihood Amir, Panagiotis Charalampopoulos, Costas S. Iliopoulos, Solon P. Pissis, and Jakub Radoszewski. Longest common factor after one edit operation. In *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.

4 Amihood Amir, Panagiotis Charalampopoulos, Solon P. Pissis, and Jakub Radoszewski. Dynamic and internal longest common substring. *Algorithmica*, 82(12):3707–3743, 2020. doi:10.1007/S00453-020-00744-0.

5 Alberto Apostolico and Andrzej Ehrenfeucht. Efficient detection of quasiperiodicities in strings. *Technical Report 90.5, The Leonadro Fibonacci Institute, Trento, Italy*, 1990.

6 Alberto Apostolico and Andrzej Ehrenfeucht. Efficient detection of quasiperiodicities in strings. *Theor. Comput. Sci.*, 119(2):247–265, 1993. doi:10.1016/0304-3975(93)90159-Q.

7 Alberto Apostolico, Martin Farach, and Costas S. Iliopoulos. Optimal superprimitivity testing for strings. *Inf. Process. Lett.*, 39(1):17–20, 1991. doi:10.1016/0020-0190(91)90056-N.

8 Michael A. Bender and Martin Farach-Colton. The LCA problem revisited. In *LATIN 2000: Theoretical Informatics, 4th Latin American Symposium, Punta del Este, Uruguay, April 10-14, 2000, Proceedings*, volume 1776 of *Lecture Notes in Computer Science*, pages 88–94. Springer, 2000. doi:10.1007/10719839_9.

9 Dany Breslauer. An on-line string superprimitivity test. *Inf. Process. Lett.*, 44(6):345–347, 1992. doi:10.1016/0020-0190(92)90111-8.

10 Panagiotis Charalampopoulos, Pawel Gawrychowski, and Karol Pokorski. Dynamic longest common substring in polylogarithmic time. In *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.

11 Panagiotis Charalampopoulos, Tomasz Kociumaka, and Philip Wellnitz. Faster approximate pattern matching: A unified approach. *CoRR*, abs/2004.08350, 2020. arXiv:2004.08350.

12 Mitsuru Funakoshi and Takuya Mieno. Minimal unique palindromic substrings after single-character substitution. In *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.

13 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.

14 Pawel Gawrychowski, Jakub Radoszewski, and Tatiana Starikovskaya. Quasi-periodicity in streams. In *30th Annual Symposium on Combinatorial Pattern Matching, CPM 2019, June 18-20, 2019, Pisa, Italy*, volume 128 of *LIPIcs*, pages 22:1–22:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPICS.CPM.2019.22.

15 Dan Gusfield. *Algorithms on Strings, Trees, and Sequences - Computer Science and Computational Biology.* Cambridge University Press, 1997. doi:10.1017/CBO9780511574931.

16 Marek Karpinski, Wojciech Rytter, and Ayumi Shinohara. An efficient pattern-matching algorithm for strings with short descriptions. *Nord. J. Comput.*, 4(2):172–186, 1997.

17 Donald E. Knuth, James H. Morris Jr., and Vaughan R. Pratt. Fast pattern matching in strings. *SIAM J. Comput.*, 6(2):323–350, 1977. doi:10.1137/0206024.

18 Tomasz Kociumaka, Jakub Radoszewski, Wojciech Rytter, and Tomasz Walen. Internal pattern matching queries in a text and applications. *CoRR*, abs/1311.6235, 2023. doi:10.48550/arXiv.1311.6235.

19 Michael G. Main and Richard J. Lorentz. An $O(n \log n)$ algorithm for finding all repetitions in a string. *J. Algorithms*, 5(3):422–432, 1984. doi:10.1016/0196-6774(84)90021-X.

20 Neerja Mhaskar and W. F. Smyth. String covering: A survey. *Fundam. Informaticae*, 190(1):17–45, 2022. doi:10.3233/FI-222164.

21 Takuya Mieno and Mitsuru Funakoshi. Data structures for computing unique palindromes in static and non-static strings. *Algorithmica*, 2023. doi:10.1007/s00453-023-01170-8.

22 Dennis W. G. Moore and William F. Smyth. An optimal algorithm to compute all the covers of a string. *Inf. Process. Lett.*, 50(5):239–246, 1994. doi:10.1016/0020-0190(94)00045-X.

23 Yuki Urabe, Yuto Nakashima, Shunsuke Inenaga, Hideo Bannai, and Masayuki Takeda. Longest Lyndon substring after edit. In *Annual Symposium on Combinatorial Pattern Matching, CPM 2018, July 2-4, 2018 - Qingdao, China*, volume 105 of *LIPIcs*, pages 19:1–19:10. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPICS.CPM.2018.19.