


# Linear-Time Computation of Cyclic Roots and Cyclic Covers of a String

Costas S. Iliopoulos  

Department of Informatics, King's College London, London, UK

Tomasz Kociumaka  

Max Planck Institute for Informatics, Saarland Informatics Campus, Saarbrücken, Germany

Jakub Radoszewski  



Institute of Informatics, University of Warsaw, Poland

Wojciech Rytter  

Institute of Informatics, University of Warsaw, Poland

Tomasz Waleń  

Institute of Informatics, University of Warsaw, Poland

Wiktor Zuba  

CWI, Amsterdam, The Netherlands

---

## Abstract

Cyclic versions of covers and roots of a string are considered in this paper. A prefix  $V$  of a string  $S$  is a *cyclic root* of  $S$  if  $S$  is a concatenation of cyclic rotations of  $V$ . A prefix  $V$  of  $S$  is a *cyclic cover* of  $S$  if the occurrences of the cyclic rotations of  $V$  cover all positions of  $S$ . We present  $\mathcal{O}(n)$ -time algorithms computing all cyclic roots (using number-theoretic tools) and all cyclic covers (using tools related to seeds) of a length- $n$  string over an integer alphabet. Our results improve upon  $\mathcal{O}(n \log \log n)$  and  $\mathcal{O}(n \log n)$  time complexities of recent algorithms of Grossi et al. (WALCOM 2023) for the respective problems and provide novel approaches to the problems. As a by-product, we obtain an optimal data structure for Internal Circular Pattern Matching queries that generalize Internal Pattern Matching and Cyclic Equivalence queries of Kociumaka et al. (SODA 2015).

**2012 ACM Subject Classification** Theory of computation  $\rightarrow$  Pattern matching

**Keywords and phrases** cyclic cover, cyclic root, circular pattern matching, internal pattern matching

**Digital Object Identifier** 10.4230/LIPIcs.CPM.2023.15

**Funding** *Jakub Radoszewski*: Supported by the Polish National Science Center, grant no. 2018/31/D/ST6/03991.

*Tomasz Waleń*: Supported by the Polish National Science Center, grant no. 2018/31/D/ST6/03991.

*Wiktor Zuba*: Supported by the Netherlands Organisation for Scientific Research (NWO) through Gravitation-grant NETWORKS-024.002.003.

## 1 Introduction

Cyclic strings have many real-world applications, such as in bioinformatics [2, 3, 17, 19] and image processing [1, 27, 28, 29]. In particular, they are used for detecting DNA viruses with circular structures [30, 31]. In particular, cyclic strings were studied in the context of circular pattern matching [10, 14, 23, 24].

In this paper, we investigate the complexity of two problems related to cyclic strings. The first one is a cyclic variant of the problem of computing the *roots* of a string  $S$ , i.e., strings  $U$  such that  $S = U^k$  for some integer  $k$ . The second one is a cyclic variant of the problem of computing the *covers* of a string  $S$ , i.e., strings  $C$  whose occurrences cover the



© Costas S. Iliopoulos, Tomasz Kociumaka, Jakub Radoszewski, Wojciech Rytter, Tomasz Waleń, and Wiktor Zuba;

licensed under Creative Commons License CC-BY 4.0

34th Annual Symposium on Combinatorial Pattern Matching (CPM 2023).

Editors: Laurent Bulteau and Zsuzsanna Lipták; Article No. 15; pp. 15:1–15:15

Leibniz International Proceedings in Informatics



Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

## 15:2 Linear-Time Computation of Cyclic Roots and Cyclic Covers of a String

whole string  $S$ . The standard roots of a string can be easily computed in linear time using a folklore algorithm. Moore and Smyth [25, 26] gave a linear-time algorithm computing all standard covers of a string. However, the cyclic versions of these problems are more difficult.

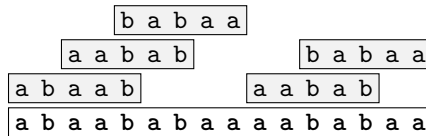
We say that a string  $V$  is a (*cyclic*) *rotation* of a string  $U$  if there exist strings  $X$  and  $Y$  such that  $U = XY$  and  $V = YX$ . A string  $U$  has a *circular occurrence* in a string  $T$  at position  $i$  if a rotation of  $U$  has a (standard) occurrence in  $T$  at position  $i$ .<sup>1</sup> By  $CircOcc(U, T)$  we denote the set of circular occurrences of  $U$  in  $T$ . Moreover, we denote

$$Covered(U, T) = \bigcup_{i \in CircOcc(U, T)} [i..i + |U|).$$

► **Definition 1.** A string  $U$  is a cyclic cover of a string  $S$  if  $Covered(U, S) = [0..|S|)$ . A string  $U$  is a cyclic root of a string  $S$  if  $S = U_1 \cdots U_k$ , where each  $U_i$  is a cyclic shift of  $U$ .

Note that if  $U$  is a cyclic root (cyclic cover) of  $S$ , then the prefix  $S[0..|U|)$  is also a cyclic root (cyclic cover, respectively) of  $S$ .

► **Example 2.** The lengths of the cyclic roots of the Thue–Morse word `abbabaabbaabba` are 2, 4, 8, 16.



■ **Figure 1** The string `abaab` is a cyclic cover of the string `abaababaaaababaa`.

► **Example 3.** The string `ab` is a cyclic cover of each Fibonacci string of length at least 2, e.g., of the string  $Fib_5 = \text{abaababa}$ . (See [13] for a definition of Fibonacci strings and their properties.) However, it is not a cyclic root of any Fibonacci string longer than 2. Another example of a cyclic cover of a string is illustrated in Figure 1.

We consider the following problems.

CYCLICROOTS

**Input:** A string  $S$  of length  $n$ .

**Output:** The lengths of all cyclic roots of  $S$ .

CYCLICCOVERS

**Input:** A string  $S$  of length  $n$ .

**Output:** The lengths of all cyclic covers of  $S$ .

### Our results

We show linear-time algorithms for both problems. We assume the word-RAM model of computation and that the string  $S$  is over an integer alphabet  $\{0, \dots, n^{O(1)}\}$ .

<sup>1</sup> We assume that the positions of a string  $T$  are numbered 0 through  $|T| - 1$ , where  $|T|$  is the length of  $T$ .

## Previous results

Recently, Grossi et al. [16] presented an  $\mathcal{O}(n \log \log n)$ -time algorithm for CYCLICROOTS (named cyclic factorization there) and an  $\mathcal{O}(n \log n)$ -time algorithm for CYCLICCOVERS.

► **Remark 4.** A completely different problem of covering a *cyclic string with a standard string cover* was considered in [11, 12]. Another different problem also known under the name “cyclic covers”, related to the shortest superstring problem, was considered in [4, 5, 6].

## Our approach

In the case of cyclic covers, we use a recursive algorithm whose general structure partially resembles the structure of the linear-time algorithm for computing seeds from [21]. For this, we need to explore combinatorics of circular occurrences, which is different from that of standard occurrences. The “working horse” of the algorithm (non-recursive parts) is the computation of long cyclic covers, which is based on an efficient implementation of internal circular pattern matching queries. Such queries require to find all circular occurrences of one substring of a text in another substring; the set of occurrences can be represented compactly if the ratio of lengths of the two strings is constant. An auxiliary contribution of our paper is an optimal implementation of these queries (constant-time after linear-time preprocessing).

Also in the case of cyclic roots we use internal queries on strings. Our algorithm is based on number-theoretic tools and fast internal queries for cyclic equivalence, which ask if two substrings of a given string are rotations of each other [20, 22].

## Notations

For a string  $S$ , by  $S[0], \dots, S[|S| - 1]$  we denote its respective letters. By  $S[i..j]$  we denote a substring  $S[i] \cdots S[j - 1]$ ; similarly, we define substrings  $S[i..j]$ ,  $S(i..j)$  and  $S(i..j)$ . We say that  $p$  is a period of a string  $S$  if  $S[i] = S[i + p]$  holds for all  $i \in [0..|S| - p]$ . By  $\text{per}(S)$ , we denote the smallest period of  $S$ , called *the period* of  $S$ . For a string  $S$  and integer  $x \in [0..|S|)$ , by  $\text{rot}_x(S)$  we denote the rotation  $S[x..|S|) \cdot S[0..x)$  of  $S$  obtained from  $S$  by moving the prefix of  $S$  of length  $x$  to the end.

A length- $m$  string  $P$  has an *occurrence* in a string  $T$  at position  $i$  if  $T[i..i + m) = P$ . By  $\text{Occ}(P, T)$  we denote the set of starting positions of occurrences of  $P$  in  $T$ .

## 2 Internal Circular Pattern Matching and CyclicCovers in $\mathcal{O}(n \log n)$ Time

We introduce the following generalization of Internal Pattern Matching queries from [22].

SIMPLE INTERNAL CIRCULAR PATTERN MATCHING QUERIES (SIMPLE INTERNALCPM)

**Input:** A string  $S$  of length  $n$ .

**Queries:** Given two substrings  $P$  and  $T$  of  $S$  such that  $|T| \leq 2|P|$ , report the leftmost and the rightmost circular occurrence of  $P$  in  $T$ .

► **Remark 5.** If we know how to compute the leftmost circular occurrence, then the rightmost circular occurrence can be computed analogously (it suffices to reverse the strings).

The theorem below can be obtained using the methods from [7, 8]. We give its proof in Section 6.

► **Theorem 6.** *The SIMPLE INTERNALCPM queries can be answered in  $\mathcal{O}(1)$  time after  $\mathcal{O}(n)$ -time preprocessing.*

## 15:4 Linear-Time Computation of Cyclic Roots and Cyclic Covers of a String

► **Remark 7.** In Section 6, we obtain a version of the queries in which a constant-sized representation of *all* circular occurrences is computed in constant time (still if  $|T| \leq 2|P|$ ).

Below, we apply SIMPLE INTERNALCPM queries to a version of the CYCLICCOVERS problem that is used in our  $\mathcal{O}(n)$ -time algorithm for CYCLICCOVERS. The following lemma generalizes [16, Lemma 13]; in this section, it will be applied in a simpler setting.

► **Lemma 8.** *After  $\mathcal{O}(n)$ -time preprocessing of a string  $S$  of length  $n$ , for any substrings  $C$  and  $W$  of  $S$ , we can test if  $C$  is a cyclic cover of  $W$  in  $\mathcal{O}(|W|/|C|)$  time.*

**Proof.** Let  $p = |C|$  and  $m = |W|$ . Consider substrings  $V_i$  equal to  $W[ip..(i+2)p]$  for  $i \in [0.. \lfloor m/p \rfloor - 1]$  and  $W[ip..m]$  for  $i = \lfloor m/p \rfloor - 1$ . For each substring  $V_i$ , we use a SIMPLE INTERNALCPM query to compute the leftmost and the rightmost circular occurrence of  $C$ . Then, we check whether these occurrences, interpreted as occurrences in  $W$ , collectively cover all positions in  $W$ . The time complexity is  $\mathcal{O}(m/p)$  after the preprocessing of Theorem 6. ◀

Lemma 8 implies a simple  $\mathcal{O}(n \log n)$ -time algorithm for the CYCLICCOVERS problem.

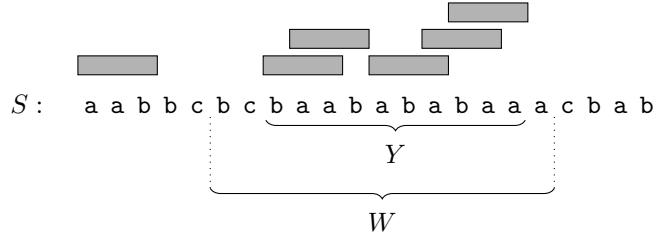
► **Corollary 9.** *The CYCLICCOVERS problem can be solved in  $\mathcal{O}(n \log n)$  time.*

**Proof.** We apply Lemma 8 for  $W = S$  iterating with  $C$  over all non-empty prefixes of  $S$ . The total time complexity is  $\mathcal{O}(n + \sum_{i=1}^n \frac{n}{i}) = \mathcal{O}(n \log n)$ . ◀

### 3 Quasi-Covers

We reduce our problem to the computation of the substrings called *quasi-covers*; see Figure 2.

► **Definition 10.** *A string  $V$  is a quasi-cover of a substring  $W$  of the string  $S$  if  $V$  is a prefix of  $S$  and a cyclic cover of a substring  $Y = W[i..j]$  such that  $i < |V|$  and  $j > |W| - |V|$ .*



■ **Figure 2** Example of a quasi-cover **aab** of the substring  $W$  of  $S$ . By definition,  $V$  is a prefix of the whole string  $S$ . Observe that **aab** is not a cyclic cover of  $W$ .

Henceforth, we fix the string  $S$  and consider quasi-covers of its substrings. Let  $W$  be a substring and  $I$  be an interval,  $I \subseteq [1..|W|]$ . We denote by  $\text{Q-COVERS}_I(W)$  the set of all lengths of quasi-covers of substring  $W$  with lengths in  $I$ . Furthermore, for  $k \in [1..|W|]$  we denote

$$\begin{aligned} \text{Q-COVERS}(W) &= \text{Q-COVERS}_{[1..|W|]}(W), & \text{Q-COVERS}_{\leq k}(W) &= \text{Q-COVERS}_{[1..k]}(W), \\ \text{Q-COVERS}_{>k}(W) &= \text{Q-COVERS}_{(k..|W|]}(W), & \text{Q-COVERS}_k(W) &= \text{Q-COVERS}_{[k..k]}(W). \end{aligned}$$

A prefix of  $S$  is its cyclic cover if and only if it is a quasi-cover of  $S$  and a rotation of a suffix of  $S$ . We use the following queries (generalized by SIMPLE INTERNALCPM queries):

## CYCLIC EQUIVALENCE QUERIES (CYCEQ)

**Input:** A string  $S$  of length  $n$ .**Queries:** Given two substrings  $U$  and  $V$  of  $S$ , check if  $V$  is a rotation of  $U$ .

► **Theorem 11** ([20, 22]). *The CYCEQ queries can be answered in  $\mathcal{O}(1)$  time after  $\mathcal{O}(n)$ -time preprocessing.*

Using CYCEQ queries, we can compute in  $\mathcal{O}(n)$  time all prefixes which are rotations of the corresponding suffixes of the string. This yields the following observation.

► **Observation 12.** *The CYCLICCOVERS problem for a string  $S$  reduces in linear time to the computation of Q-COVERS( $S$ ).*

We will later show how the set Q-COVERS( $S$ ) can be computed based on recursive calls to Q-COVERS( $W$ ) for substrings  $W$  of  $S$ .

### 3.1 Quasi-Covers and Substring Complexity

The substring complexity of a length- $m$  string  $W$  is a function that maps each length  $k \in [1..m]$  to the number  $|\text{SUB}_k(W)|$  of distinct length- $k$  substrings of  $W$ . We further define  $\beta_k(W) = |\text{SUB}_k(W)| + k - 1$ . The term  $k - 1$  is added because the sequence  $(|\text{SUB}_k(W)|)_{k=1}^m$  does not need to be monotone in general; the resulting sequence  $(\beta_k(W))_{k=1}^m$  is now non-decreasing and its monotonicity will be useful later. For a string family  $\mathcal{S}$ , let us denote by  $\|\mathcal{S}\|$  the sum of lengths of strings in  $\mathcal{S}$ .

► **Observation 13.** *Let  $V$  be a quasi-cover of a substring  $W$  of  $S$ . If a substring  $W'$  of  $W$  satisfies  $|W'| \geq 2|V| - 1$ , then  $V$  is a quasi-cover of  $W'$ .*

► **Lemma 14.** *Given a length- $m$  substring  $W$  and an integer  $k \in [1..m]$ , we can compute in  $\mathcal{O}(m)$  time a family  $\mathbf{G}_k(W)$  of substrings of  $W$  such that  $\|\mathbf{G}_k(W)\| \leq \beta_k(W)$  and*

$$\text{Q-COVERS}_{\leq \lceil k/4 \rceil}(W) = \bigcap_{W' \in \mathbf{G}_k(W)} \text{Q-COVERS}_{\leq \lceil k/4 \rceil}(W').$$

**Proof.** It was shown in [21] that one can construct in linear time a string family, denoted in [21] as  $\text{COMPR}_t$ , such that  $\|\text{COMPR}_t\| \leq \beta_{2t-1}(W)$  and the strings in  $\text{COMPR}_t$  contain all length- $t$  substrings of  $W$ . First, we reformulate the corresponding fact from [21] taking  $\mathbf{G}_k(W) = \text{COMPR}_{\lceil k/2 \rceil}$ .

► **Claim 15** ([21, Lemma 5.4, proof of Lemma 5.3 and proof of Theorem 9 (“Computing  $S$ ”)]). Given  $k \in [1..m]$ , we can compute in  $\mathcal{O}(m)$  time a string family  $\mathbf{G}_k(W)$  such that

$$\|\mathbf{G}_k(W)\| \leq \beta_k(W) \quad \text{and} \quad \text{SUB}_{\lceil k/2 \rceil}(W) = \bigcup_{W' \in \mathbf{G}_k(W)} \text{SUB}_{\lceil k/2 \rceil}(W').$$

The next claim turns out to be similar to [21, Lemma 2.2].

► **Claim 16.** If  $t \in [1..m]$ , then

$$\text{Q-COVERS}_{\leq \lceil t/2 \rceil}(W) = \bigcap_{W' \in \text{SUB}_t(W)} \text{Q-COVERS}_{\leq \lceil t/2 \rceil}(W').$$

## 15:6 Linear-Time Computation of Cyclic Roots and Cyclic Covers of a String

Proof. We prove two inclusions separately.

( $\subseteq$ ) If  $V$  is a quasi-cover of  $W$  of length at most  $\lceil t/2 \rceil$  and  $W' \in \text{SUB}_t(W)$ , then Observation 13 implies that  $V$  is a quasi-cover of  $W'$ .

( $\supseteq$ ) Assume that  $V$  is a quasi-cover of all  $W' \in \text{SUB}_t(W)$  and  $|V| = \ell \leq \lceil t/2 \rceil$ . Consider a position  $i \in [\ell - 1 .. m - \ell]$  and a substring  $W' = W(i - \ell .. i + \ell)$ . Note that  $V$  is a quasi-cover of  $W'$  (this follows from Observation 13 because  $W'$  is a substring of some length- $t$  substring of  $W$ ). Thus, there is a circular occurrence of  $V$  in  $W$  that covers the middle position of  $W'$ . Interpreted as a circular occurrence of  $V$  in  $W$ , it covers position  $i$  of  $W$ .  $\triangleleft$

The thesis follows directly from the two claims above, taking  $t = \lceil k/2 \rceil$  in the second claim.  $\blacktriangleleft$

► **Lemma 17.** *If a string  $W$  has a quasi-cover  $V$  of length  $|V| \geq 2k$ , then*

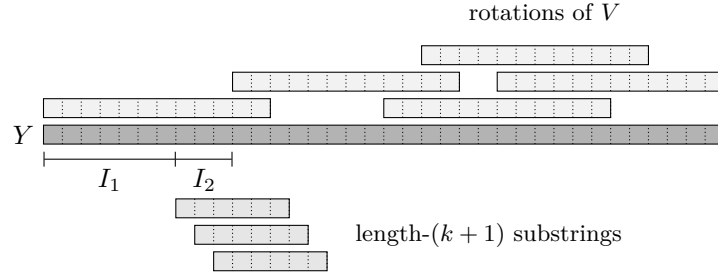
$$|\text{SUB}_{k+1}(W)| \leq \frac{1}{2}|W| + \frac{3}{2}|V|.$$

**Proof.** First, we show the following claim (cf. Figure 3).

► **Claim 18.** *If a string  $Y$  has a cyclic cover  $V$  of length  $|V| \geq 2k$ , then  $|\text{SUB}_{k+1}(Y)| \leq \frac{1}{2}(|Y| + |V|)$ .*

Proof. We denote by  $\text{CSUB}_{k+1}(V)$  the set of distinct length- $(k+1)$  substrings of all rotations of  $V$ ; note that  $|\text{CSUB}_{k+1}(V)| \leq |V|$ .

For each  $i \in \text{CircOcc}(V, Y)$ , let us mark positions  $j \in [i .. i + |V| - k]$ ; observe that if a position  $j$  is marked, then  $Y[j .. j + k] \in \text{CSUB}_{k+1}(V)$ .



■ **Figure 3** A string  $V$  is a cyclic cover of a string  $Y$  and  $|V| \geq 2k$ . Each length- $(k+1)$  substring starting in  $I_1$  belongs to  $\text{CSUB}_{k+1}(V)$ , but length- $(k+1)$  substrings starting in  $I_2$  do not need to belong to  $\text{CSUB}_{k+1}(V)$ . The interval  $I_1$  of marked positions is of size at least  $k$ , whereas the interval  $|I_2|$  of unmarked positions is of size at most  $k$ .

Let  $Y'$  be the prefix of  $Y$  of length  $|Y| - |V|$ . We partition  $Y'$  into inclusion-wise maximal intervals of marked positions and inclusion-wise maximal intervals of unmarked positions. Each interval  $I_2$  of unmarked positions is preceded by an interval  $I_1$  of marked positions, where  $|I_1| \geq |V| - k \geq k \geq |I_2|$  (otherwise,  $V$  would not be a cyclic cover of  $Y$ ).

Hence, at most half of positions of  $Y'$  are unmarked, which is  $(|Y| - |V|)/2$ . Each length- $(k+1)$  substring starting at marked position belongs to  $\text{CSUB}_{k+1}(V)$ . Hence,  $|\text{SUB}_{k+1}(Y)| \leq (|Y| - |V|)/2 + |\text{CSUB}_{k+1}(V)| \leq (|Y| + |V|)/2$ .  $\triangleleft$

If  $V$  is a quasi-cover of  $W$  then  $V$  is a cyclic cover of  $Y = W[i .. j]$  with  $i < |V|$  and  $j > m - |V|$ . We have, due to inequality  $\frac{1}{2}(|W| - |Y|) < |V|$ ,

$$|\text{SUB}_{k+1}(W)| \leq |\text{SUB}_{k+1}(Y)| + |W| - |Y| < |\text{SUB}_{k+1}(Y)| + \frac{1}{2}(|W| - |Y|) + |V|.$$

Now, Claim 18 implies  $|\text{SUB}_{k+1}(Y)| \leq \frac{1}{2}(|Y| + |V|)$  and thus  $|\text{SUB}_{k+1}(W)| \leq \frac{1}{2}|W| + \frac{3}{2}|V|$ .  $\blacktriangleleft$

► **Example 19.** Let  $\{a_1, a_2, \dots, a_{2k-1}\}$ ,  $\{b_1, b_2, \dots, b_{2k}\}$ ,  $\{c_1, c_2, \dots, c_{2k-1}\}$  be disjoint sets, and

$$X = a_1 a_2 \dots a_{2k-1}, \quad V_1 = b_1 b_2 \dots b_k, \quad V_2 = b_{k+1} b_{k+2} \dots b_{2k}, \quad Y = c_1 c_2 \dots c_{2k-1}.$$

Then  $V = V_1 V_2$  is a quasi-cover of  $W = X V_1 V_2 V_2 V_1 Y$ , with  $|V| = 2k$  and  $|W| = 8k - 2$ . All length- $(k+1)$  substrings of  $W$  are different. Hence:

$$|\text{SUB}_{k+1}(W)| = |W| - k = \frac{1}{2}|W| + \frac{3}{2}|V| - o(|W|).$$

We use the following crucial property of quasi-covers.

► **Lemma 20 (Work-Reduction Lemma).** *For a length- $m$  substring  $W$  and  $k \in [0..m)$ , if  $\beta_{k+1}(W) > \frac{5}{6}m$ , then*

$$\text{Q-COVERS}_{[2k.. \lfloor m/6 \rfloor]}(W) = \emptyset.$$

**Proof.** The proof is by contradiction. Suppose that  $V \in \text{Q-COVERS}_{[2k.. \lfloor m/6 \rfloor]}(W)$  and  $V$  is a cyclic cover of  $Y = W[i..j]$  with  $i < |V|$  and  $j > m - |V|$ . Due to Lemma 17 and inequality  $k \leq |V|/2$ ,

$$|\text{SUB}_{k+1}(W)| + k \leq \frac{1}{2}|W| + \frac{3}{2}|V| + \frac{1}{2}|V| = \frac{1}{2}|W| + 2|V| \leq \frac{1}{2}m + 2 \cdot \frac{m}{6} = \frac{5}{6}m.$$

This contradicts our assumption that  $\beta_{k+1}(W) = |\text{SUB}_{k+1}(W)| + k > \frac{5}{6}m$ . ◀

## 4 Solution to CyclicCovers Problem

Our algorithm is recursive; its non-recursive parts correspond to (simple) fast computation of length-limited cyclic covers. We say that an interval  $I = [a..b]$  of positive integers is *balanced* if  $b = \mathcal{O}(a)$ .

► **Lemma 21.** *After  $\mathcal{O}(n)$ -time preprocessing, for a balanced interval  $I = [a..b]$  and a length- $m$  substring  $W$ , the set  $\text{Q-COVERS}_I(W)$  can be computed in  $\mathcal{O}(m)$  time.*

**Proof.** We consider each length  $\ell \in I$  separately. Let  $C = S[0..\ell)$ . We use two SIMPLE INTERNALCPM queries to check if  $C$  has a circular occurrence starting within the first  $\ell$  positions of  $W$  and a circular occurrence ending within the last  $\ell$  positions of  $W$ . If any of these two conditions does not hold,  $C$  is not a quasi-cover of  $W$ . Otherwise, we use Lemma 8 to check if  $C$  is a cyclic cover of the substring of  $W$  spanned by the first and the last circular occurrence of  $C$  in  $W$  that were discovered in the previous step. The total time complexity is  $\mathcal{O}(\sum_{i \in I} \frac{m}{i}) = \mathcal{O}(\sum_{i \in I} \frac{m}{a}) = \mathcal{O}(m \cdot |I|/a) = \mathcal{O}(m \cdot b/a) = \mathcal{O}(m)$ , after  $\mathcal{O}(n)$ -time preprocessing in Theorem 6 and Lemma 8. ◀

Our solution is based on Lemmas 14, 20, and 21. We use a recursive approach that was initially developed for seeds computation; see [21].

► **Theorem 22.** *The CYCLICCOVERS problem can be solved in  $\mathcal{O}(n)$  time.*

**Proof.** We run the recursive function `ComputeQuasiCovers` (Algorithm 1) initially for  $W = S$ .

**Correctness.** In the base case, where  $\beta_1(W) > \frac{5}{6}m$ , there are more than  $\frac{5}{6}m$  different letters in  $W$ , and then Lemma 20 implies  $\text{Q-COVERS}_{\leq \lfloor m/6 \rfloor}(W) = \emptyset$ .

In the recursive step, we reduce the computation of quasi-covers to the ones with lengths in two balanced intervals,  $J_1 = (\lceil k/4 \rceil .. 2k)$  and  $J_2 = (\lfloor m/6 \rfloor .. m)$ , and the ones (the set  $Q$ ) with sufficiently small lengths (at most  $\lceil k/4 \rceil$ ). By Lemmas 14 and 20, the algorithm returns precisely the set  $\text{Q-COVERS}(W)$ .

**Algorithm 1** ComputeQuasiCovers( $W$ ).

---

**Input:** A substring  $W$  of length  $m$ .  
**Output:** The set Q-COVERS( $W$ ) of lengths of quasi-covers of  $W$ .  
 Compute  $\beta_\ell(W)$  for all  $\ell \in [1..m]$   
**if**  $\beta_1(W) > \frac{5}{6}m$  **then** ▶ see Lemma 20  
     **return** Q-COVERS $_{\lfloor m/6 \rfloor .. m}(W)$  ▶  $\lfloor m/6 \rfloor .. m$  is a balanced interval  
 $k := \max \{ \ell \in [1..m] : \beta_\ell(W) \leq \frac{5}{6}m \}$   
 Let  $\mathbf{G}_k(W)$  be the set of fragments as in Lemma 14  
**foreach** string  $W' \in \mathbf{G}_k(W)$  **do**  
      $Q_{W'} := \text{ComputeQuasiCovers}(W')$  ▶ Recursive call  
      $Q := \bigcap_{W' \in \mathbf{G}_k(W)} Q_{W'} \cap [1.. \lceil k/4 \rceil]$  ▶  $Q = \text{Q-COVERS}_{\leq \lceil k/4 \rceil}(W)$  (Lemma 14)  
      $J_1 := (\lceil \frac{k}{4} \rceil .. 2k), J_2 := (\lfloor \frac{m}{6} \rfloor .. m)$  ▶  $J_1, J_2$  are balanced intervals  
     **return**  $Q \cup \text{Q-COVERS}_{J_1}(W) \cup \text{Q-COVERS}_{J_2}(W)$  ▶  $\text{Q-COVERS}_{[2k.. \lfloor m/6 \rfloor]}(W) = \emptyset$

---

**Complexity.** To bound the running time, denote by  $T(m)$  the maximum number of operations performed by the algorithm for a substring  $W$  of length  $m$ . The sequence  $\beta_i(W)$  for a length- $m$  substring  $W$  of  $S$  can be computed in  $\mathcal{O}(m)$  time [21, Lemma 5.1]. Due to Lemmas 14 and 21,

$$T(m) = \mathcal{O}(m) + \sum_i T(m_i), \quad \text{where } \sum_i m_i \leq \frac{5}{6}m.$$

This recurrence yields  $T(m) = \mathcal{O}(m)$ .

Due to Observation 12, the CYCLICCOVERS problem can be reduced in linear time to the computation of all quasi-covers. Finally, Lemma 21 requires  $\mathcal{O}(n)$ -time preprocessing of  $S$ . This completes the proof. ◀

## 5

 Solution to CyclicRoots Problem

We denote by  $\sigma_0(n) = \sum_{p|n} 1$  the number of divisors of  $n$  and by  $\sigma_1(n) = \sum_{p|n} p$  the sum of divisors of  $n$ . We use the following known estimations:  $\sigma_0(n) = 2^{\mathcal{O}(\log n / \log \log n)}$  [18, §18.1] and  $\sigma_1(n) = \mathcal{O}(n \log \log n)$  [18, §22.9]. They directly imply the following fact.

▶ **Fact 23.**

- $\sigma_0(n) = o(\sqrt{n} / \log n)$  and  $\log \sigma_0(n) = \mathcal{O}(\log n / \log \log n)$
- $\sigma_1(n) = \sum_{p|n} \frac{n}{p} = \mathcal{O}(n \log \log n)$

Using CYCEQ queries (Theorem 11), we derive the following subroutine:

▶ **Observation 24.** After linear-time preprocessing of a string  $S$ , we can test if  $S[0..p]$  is a cyclic root of a substring  $W$  of  $S$  in  $\mathcal{O}(|W|/p)$  time.

In particular, in [16] the CYCLICROOTS problem was solved in  $\mathcal{O}(\sigma_1(n)) = \mathcal{O}(n \log \log n)$  time (cf. Fact 23) by using  $\frac{n}{p}$  CYCEQ queries for each divisor  $p$  of  $n$ .

Let us now develop an  $\mathcal{O}(n)$ -time solution. We reduce testing if  $S[0..p]$  is a cyclic root of the whole text to testing if  $S[0..p]$  is a cyclic root of each substring  $F$  in a suitably chosen family  $\mathcal{F}$  of substrings.



The intuition behind this improvement is as follows. It turns out that the asymptotic upper bound on  $\sigma_1(n)$  significantly depends on a few largest divisors. In the  $\mathcal{O}(n \log \log n)$ -time algorithm, this corresponds to the smallest lengths  $p$  of the candidate cyclic root. Hence, for small  $p$ , we will adopt a different approach.

The factorization of  $S$  into length- $q$  substrings, for  $q \mid n$ , will be called the  $q$ -factorization.

► **Observation 25.** *If  $S$  has a cyclic root of length  $p$ , then its  $(k \cdot p)$ -factorization  $\mathcal{F}$  contains at most  $p^k$  distinct substrings, consequently the number of distinct factors in  $\mathcal{F}$  is at most  $\min\left(p^k, \frac{n}{k \cdot p}\right)$ .*

Thus, if the number of different factors in the  $(k \cdot p)$ -factorization is greater than  $p^k$ , then we know that  $S$  does not have a cyclic root of length  $p$ .

Otherwise, if  $k$  is small enough, the number of different substrings in the  $(k \cdot p)$ -factorization will be smaller than  $n/(k \cdot p)$ , and we can check each of them using CYCEQ queries in  $\mathcal{O}(k)$  time. On the other hand, if  $k$  is large enough, then the  $\mathcal{O}(n/(k \cdot p))$  work spent on computing the factorization will be much less than  $\mathcal{O}(n/p)$ .

■ **Algorithm 2** `CyclicRoot( $S, p$ )`: Does  $S$  have a cyclic root of length  $p \mid n$ ?

---

$k := \max(\lfloor \frac{1}{2} \log_p n \rfloor, 1)$

Let  $S = S_1 S_2$ , where  $|S_2| = n \bmod (k \cdot p)$

$\mathcal{F} :=$  all distinct factors in the  $(k \cdot p)$ -factorization of  $S_1$    ►  $\mathcal{O}(n/(k \cdot p))$  time [15]

if  $|\mathcal{F}| > p^k$  **then return** *NO*   ► Observation 25

if  $|S_2| > 0$  **then**  $\mathcal{F} := \mathcal{F} \cup \{S_2\}$    ►  $|\mathcal{F}| = \mathcal{O}\left(\min\left(p^k, \frac{n}{k \cdot p}\right)\right)$

**foreach**  $F \in \mathcal{F}$  **do**   ►  $\mathcal{O}(k \cdot |\mathcal{F}|) = \mathcal{O}(\sqrt{n} \log n)$  time

**if**  $S[0..p]$  *is not a cyclic root of*  $F$  **then return** *NO*   ► Observation 24

**return** *YES*

---

► **Theorem 26.** *The CYCLICROOTS problem can be solved in  $\mathcal{O}(n)$  time.*

**Proof.** We use Algorithm 2. After  $\mathcal{O}(n)$ -time preprocessing, all different substrings in  $\mathcal{F}$  can be found in  $\mathcal{O}(n/(k \cdot p))$  time using deterministic substring hashing [15]. By Observation 24, after  $\mathcal{O}(n)$ -time preprocessing, we can test in  $\mathcal{O}(k)$  time for each  $F \in \mathcal{F}$  if  $S[0..p]$  is its cyclic root; this sums up to  $\mathcal{O}(k \cdot \min(\frac{n}{k \cdot p}, p^k))$  time. For  $k = \max(\lfloor \frac{1}{2} \log_p n \rfloor, 1)$ , we have

$$\frac{n}{k \cdot p} = \mathcal{O}\left(\frac{n \log p}{p \log n}\right) \quad \text{and} \quad k \cdot \min\left(p^k, \frac{n}{k \cdot p}\right) = \mathcal{O}\left(p^{\frac{1}{2} \log_p n} \cdot \log_p n + \min\left(p, \frac{n}{p}\right)\right) = \mathcal{O}(\sqrt{n} \log n).$$

Thus, after  $\mathcal{O}(n)$ -time preprocessing, all calls to the algorithm `CyclicRoot( $S, p$ )` for all divisors  $p$  of  $n$  work in total time

$$\mathcal{O}(A(n) + B(n)), \quad \text{where} \quad A(n) = \sum_{p \mid n} \frac{n \log p}{p \log n} \quad \text{and} \quad B(n) = \sum_{p \mid n} \sqrt{n} \cdot \log n.$$

**Estimating  $B(n)$ .** By Fact 23, we have

$$B(n) = \sqrt{n} \log n \cdot \sum_{p \mid n} 1 = \sqrt{n} \log n \cdot \sigma_0(n) = o(n).$$

## 15:10 Linear-Time Computation of Cyclic Roots and Cyclic Covers of a String

**Estimating  $A(n)$ .** We partition the underlying sum into elements that do not exceed  $\sigma_0(n)$  and the remaining elements. The former is bounded from above, due to Fact 23, as:

$$\sum_{\substack{p|n \\ p \leq \sigma_0(n)}} \frac{n \log p}{p \log n} \leq \sum_{p|n} \frac{n \log \sigma_0(n)}{p \log n} = \frac{\log \sigma_0(n)}{\log n} \sum_{p|n} \frac{n}{p} = \mathcal{O}\left(\frac{1}{\log \log n} \cdot \sigma_1(n)\right) = \mathcal{O}(n).$$

The latter sum is bounded from above by:

$$\sum_{\substack{p|n \\ p > \sigma_0(n)}} \frac{n \log p}{p \log n} \leq \sum_{p|n} \frac{n}{\sigma_0(n)} = \frac{n}{\sigma_0(n)} \sum_{p|n} 1 = \frac{n}{\sigma_0(n)} \cdot \sigma_0(n) = n.$$

This concludes the complexity analysis of the algorithm. ◀

## 6 InternalCPM via PILLAR Model

### 6.1 PILLAR Model

We use the so-called PILLAR model that was introduced in [9]. In this model, we assume that the following primitive queries can be performed efficiently, where the argument strings are represented as substrings of strings in a given collection  $\mathcal{X}$ :

- **Extract**( $U, \ell, r$ ): Retrieve the substring  $U[\ell..r]$ .
- **LCP**( $U, V$ ), **LCP<sub>R</sub>**( $U, V$ ): Find the length of the longest common prefix/suffix of  $U$  and  $V$ .
- **IPM**( $U, V$ ): Assuming that  $|V| < 2|U|$ , compute the starting positions of all exact occurrences of  $U$  in  $V$ , expressed as an arithmetic sequence. If the sequence has at least three terms, its difference equals  $\text{per}(U)$ .
- **Access**( $U, i$ ): Retrieve the letter  $U[i]$ .
- **Length**( $U$ ): Compute the length  $|U|$  of the string  $U$ .

The runtime of algorithms in this model can be expressed in terms of the number of primitive PILLAR operations. The following result combines several known techniques to obtain constant-time implementations of all PILLAR operations in the standard setting. Efficient implementations of the PILLAR operations in other settings, including a dynamic and a compressed setting, are also known; cf. [9].

► **Theorem 27** ([9, Theorem 7.2]). *After an  $\mathcal{O}(n)$ -time preprocessing of a collection of strings of total length  $n$  over an integer alphabet, each PILLAR operation can be performed in  $\mathcal{O}(1)$  time.*

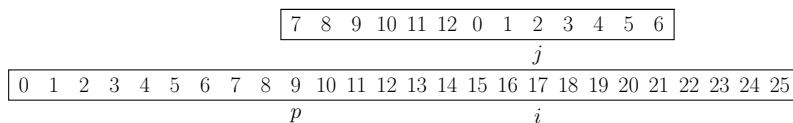
### 6.2 Interval Chains and PairMatch problem

For an integer set  $A$  and an integer  $r$ , let  $A \oplus r = \{a + r : a \in A\}$ . An *interval chain* is a set of the form  $I \cup (I \oplus q) \cup (I \oplus 2q) \cup \dots \cup (I \oplus aq)$  for an interval  $I$  and non-negative integers  $a$  and  $q$ . In particular, a single interval is an interval chain (with  $a = 0$ ).

First, we introduce an auxiliary operation **PAIRMATCH**. Denote by  $\text{PAIRMATCH}(T, P, i, j)$  the set of all circular occurrences of  $P$  in  $T$  such that position  $i$  in  $T$  is aligned with position  $j$  in  $P$  (see also Figure 4):

$$\text{PAIRMATCH}(T, P, i, j) = \{p \in (i - m..i] : T[p..p+m) = \text{rot}_x(P), i - p = (j - x) \bmod m\}.$$

In particular  $\text{PAIRMATCH}(T, P, i, 0)$  is the set of circular occurrences of  $P$  such that the leftmost position of  $P$  is aligned with position  $i$  in  $T$ .



■ **Figure 4** Let us put original numbers in positions of the pattern  $P$ ; they are moved after rotation of  $P$ . Assume that  $p = 9$  is a position of a circular occurrence of  $P$  in  $T$  such that  $p \in \text{Occ}(\text{rot}_7(P), T)$ . Then, in particular,  $p \in \text{PAIRMATCH}(T, P, 17, 2)$ . In this case,  $x = 7$  and  $i - p = 8 = (j - x) \bmod 13$ . We also have  $p \in \text{PAIRMATCH}(T, P, 15, 0)$ .

The following lemma is a consequence of [7, Lemma 10], where the PILLAR model was not used explicitly. A similar fact was shown in [16, Lemmas 5 and 6]. We include its proof for completeness.

► **Lemma 28.** *For any given  $i, j$ , the set  $\text{PAIRMATCH}(T, P, i, j)$ , represented as a union of at most two intervals, can be computed in  $\mathcal{O}(1)$  time in the PILLAR model.*

**Proof.** First we explain how to compute  $\text{PAIRMATCH}(T, P, i, 0)$ . Let  $p(i) = \text{LCP}(T[i..], P)$  and  $s(i) = \text{LCP}_R(T[.i], P)$ . If  $p(i) + s(i) \geq m$ , an interval  $[i - s(i) .. i + p(i) - m]$  of starting positions of circular occurrences of  $P$  in  $T$  is reported; otherwise the answer is an empty set.

In general  $\text{PAIRMATCH}(T, P, i, j)$  can be computed using (at most) two queries of the type  $\text{PAIRMATCH}(T, P, i', 0)$ , for  $i' = i - j$  and  $i' = i - j + m$ . A respective query is asked only if  $i' \in [0..n - m]$ . The resulting intervals need to be intersected with  $(i - m..i]$  to ensure that the circular occurrence contains position  $i$ . ◀

### 6.3 Internal CPM

The circular pattern matching problem is formally defined as follows.

CIRCULAR PATTERN MATCHING (CPM)  
**Input:** A text  $T$  of length  $n$  and a pattern  $P$  of length  $m$ .  
**Output:** All positions in  $T$  where circular occurrences of  $P$  start.

We will show an efficient solution in the PILLAR model of CPM in the case when the lengths of the pattern and of the texts are similar. The algorithm below applies the results of [7, 8]. These results considered the approximate CPM problem with  $k \geq 1$  mismatches or edits. In the proof of the following theorem, we show that they can be adapted to the case of the exact CPM problem, obtaining an even simpler algorithm. The main idea of the algorithm is illustrated in Figure 7.

► **Theorem 29.** *If  $n \leq 2m$ , the answer to the CPM problem, represented as a union of  $\mathcal{O}(1)$  interval chains, can be computed in  $\mathcal{O}(1)$  time in the PILLAR model.*

**Proof.** Let  $P = P_1P_2$ , where  $|P_1| = \lfloor m/2 \rfloor$ . Each circular occurrence of  $P$  in  $T$  implies a standard occurrence of at least one of  $P_1$  and  $P_2$  in  $T$ . Henceforth, we assume that it implies an occurrence of  $P_1$ ; the remaining case can be treated symmetrically.

Let  $A = \text{Occ}(P_1, T)$ . As  $|T| \leq 4|P_1| + 3$ , a representation of  $A$  consisting of  $\mathcal{O}(1)$  arithmetic sequences can be computed using  $\mathcal{O}(1)$  IPM queries by the so-called standard trick. We consider each of the arithmetic sequences  $B$  separately.

## 15:12 Linear-Time Computation of Cyclic Roots and Cyclic Covers of a String

**Nonperiodic case.** If an arithmetic sequence  $B$  contains at most two occurrences, then we ask a query  $\text{PAIRMATCH}(T, P, i, 0)$  for each  $i \in B$ . The resulting intervals contain positions of all circular occurrences of  $P$  in  $T$  that imply an occurrence of  $P_1$  in  $T$  at a position  $i \in B$ , and possibly some other circular occurrences of  $P$  in  $T$  (that imply an occurrence of  $P_2$ ).

**Periodic case.** Assume now that an arithmetic sequence  $B$  contains at least three elements. As already mentioned, its difference is  $q := \text{per}(P_1)$ .

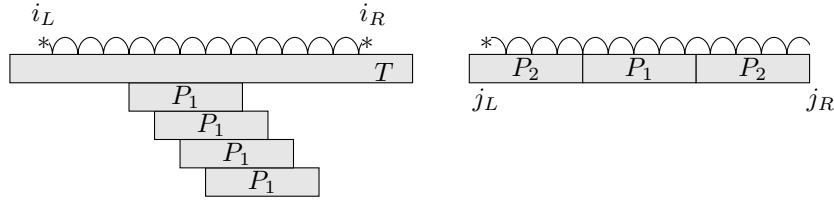
Let  $i$  be any element of  $B$ . We compute the largest index  $i_L < i$  and the smallest index  $i_R > i$  such that

$$T[i_L] \neq T[i_L + q] \text{ (or } i_L = -1), \quad T[i_R] \neq T[i_R - q] \text{ (or } i_R = |T|).$$

Let  $Q = P_2P_1P_2$  and  $j = |P_2|$ . Similarly (see Figure 5), we compute the largest index  $j_L < j$  and the smallest index  $j_R > j$  such that

$$Q[j_L] \neq Q[j_L + q] \text{ (or } j_L = -1), \quad Q[j_R] \neq Q[j_R - q] \text{ (or } j_R = |Q|).$$

The indices  $i_L, i_R, j_L, j_R$ , which can be called *misperiods*, can be computed using a constant number of LCP and LCP<sub>R</sub> queries on  $T$  and  $P$ .



■ **Figure 5** Misperiods  $i_L, i_R, j_L$ ; in this case, there is no misperiod  $j_R$ .

We consider two cases:

**Case (1).** The cyclic occurrence is an occurrence of a rotation of  $P$  that is a length- $m$  substring of  $Q(j_L .. j_R)$ ; the occurrence is contained within a substring  $T(i_L .. i_R)$  in the text. Both strings in scope are periodic with period  $q$ ; it only matters if the periods are synchronized. Let

$$X = (j_L .. j_R - m] \text{ and } Z = (i_L .. i_R - m].$$

The set  $X$  consists of the positions in  $Q$  where a rotation of  $P$  contained in  $Q(j_L .. j_R)$  starts. The set  $Z' := \{z \in Z : \exists x \in X z \equiv x \pmod{q}\}$  consists of the starting positions of circular occurrences of  $P$  contained in  $T(i_L .. i_R)$ . By the following claim, the set  $Z'$  can be computed in  $\mathcal{O}(1)$  time.

▷ **Claim 30** ([7, Lemma 7]). Let  $X$  and  $Z$  be intervals and  $q$  be a positive integer. The set  $Z' := \{z \in Z : \exists x \in X z \equiv x \pmod{q}\}$ , represented as a disjoint union of at most three interval chains, can be computed in  $\mathcal{O}(1)$  time.

**Case (2).** In this case, two misperiods, one in  $T$  and one in  $P$ , need to be synchronized. It suffices to take the union of results of a  $\text{PAIRMATCH}(T, P, i_L, |P_1| + j_L)$  query if neither of  $i_L, j_L$  equals  $-1$  and of a  $\text{PAIRMATCH}(T, P, i_R, j_R - |P_2|)$  query if  $i_R \neq |T|$  and  $j_R \neq |P|$ .

Overall, the result is a union of  $\mathcal{O}(1)$  intervals and interval chains and can be computed in  $\mathcal{O}(1)$  time in the PILLAR model using Lemma 28 and Claim 30. ◀

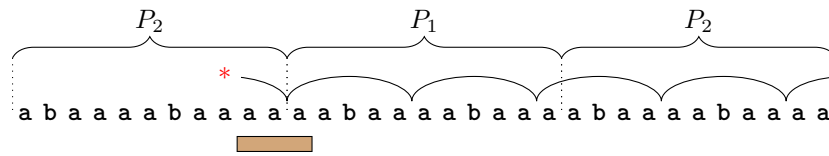


Figure 6 The interval  $X$  (shaded box) represents the starting positions of the rotations of  $P = (\text{aabaa})^4\text{aa}$  contained in  $Q(j_L..j_R) = Q(8..33)$ . Five copies of  $X$  (two of them partial) constitute the output set  $Z'$  (the shaded boxes in Figure 7).

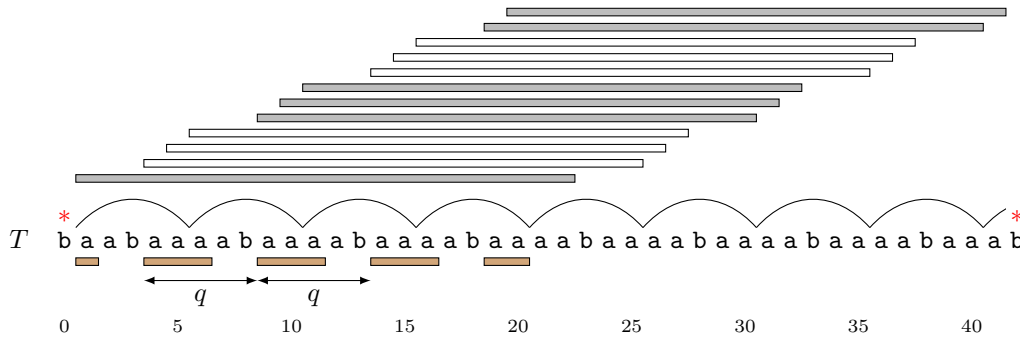


Figure 7 The starting positions of the circular occurrences of the pattern  $P = (\text{aabaa})^4\text{aa}$  in the text  $T$  form two intervals ( $[1..1]$  and  $[19..20]$ ) and one interval chain  $I, I \oplus q, I \oplus 2q$ , where  $I = [4..6]$  and  $q = 5$ .

We introduce the following generalization of IPM queries.

INTERNAL CIRCULAR PATTERN MATCHING QUERIES (INTERNALCPM)  
**Input:** A string  $S$  of length  $n$ .  
**Queries:** Given two substrings  $P$  and  $T$  of  $S$  such that  $|T| \leq 2|P|$ , report all the starting positions of all circular occurrences of  $P$  in  $T$ .

Combining Theorems 27 and 29, we obtain the following result, which generalizes Theorem 6.

► **Theorem 31.** *The answer to an INTERNALCPM query, represented as a union of  $\mathcal{O}(1)$  interval chains, can be computed in  $\mathcal{O}(1)$  time after  $\mathcal{O}(n)$ -time preprocessing.*

## 7 Final Remarks

We took a recursive approach proposed in the computation of seeds and adjusted it to the case of cyclic covers. Despite the similarity, several major changes were necessary due to circularity. We hope that such a recursive approach can be used in other problems on strings.

We also demonstrated the importance of a new tool in computations on cyclic strings: internal circular pattern matching queries. Hopefully, they could be used for other problems related to cyclic substrings.

### References

- 1 Lorraine A. K. Ayad, Carl Barton, and Solon P. Pissis. A faster and more accurate heuristic for cyclic edit distance computation. *Pattern Recognition Letters*, 88:81–87, 2017. doi:10.1016/j.patrec.2017.01.018.
- 2 Lorraine A. K. Ayad and Solon P. Pissis. MARS: improving multiple circular sequence alignment using refined sequences. *BMC Genomics*, 18(1):86, 2017. doi:10.1186/s12864-016-3477-5.

- 3 Carl Barton, Costas S. Iliopoulos, Ritu Kundu, Solon P. Pissis, Ahmad Retha, and Fatima Vayani. Accurate and efficient methods to improve multiple circular sequence alignment. In Evripidis Bampis, editor, *Experimental Algorithms, SEA 2015*, volume 9125 of *Lecture Notes in Computer Science*, pages 247–258. Springer, 2015. doi:10.1007/978-3-319-20086-6\_19.
- 4 Bastien Cazaux, Rodrigo Cánovas, and Eric Rivals. Shortest DNA cyclic cover in compressed space. In Ali Bilgin, Michael W. Marcellin, Joan Serra-Sagristà, and James A. Storer, editors, *2016 Data Compression Conference, DCC 2016*, pages 536–545. IEEE, 2016. doi:10.1109/DCC.2016.79.
- 5 Bastien Cazaux and Eric Rivals. A linear time algorithm for shortest cyclic cover of strings. *Journal of Discrete Algorithms*, 37:56–67, 2016. doi:10.1016/j.jda.2016.05.001.
- 6 Bastien Cazaux and Eric Rivals. The power of greedy algorithms for approximating Max-ATSP, Cyclic Cover, and superstrings. *Discrete Applied Mathematics*, 212:48–60, 2016. doi:10.1016/j.dam.2015.06.003.
- 7 Panagiotis Charalampopoulos, Tomasz Kociumaka, Solon P. Pissis, Jakub Radoszewski, Wojciech Rytter, Juliusz Straszłyński, Tomasz Waleń, and Wiktor Zuba. Circular pattern matching with  $k$  mismatches. *Journal of Computer and System Sciences*, 115:73–85, 2021. doi:10.1016/j.jcss.2020.07.003.
- 8 Panagiotis Charalampopoulos, Tomasz Kociumaka, Jakub Radoszewski, Solon P. Pissis, Wojciech Rytter, Tomasz Waleń, and Wiktor Zuba. Approximate circular pattern matching. In Shiri Chechik, Gonzalo Navarro, Eva Rotenberg, and Grzegorz Herman, editors, *30th Annual European Symposium on Algorithms, ESA 2022*, volume 244 of *LIPICs*, pages 35:1–35:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICs.ESA.2022.35.
- 9 Panagiotis Charalampopoulos, Tomasz Kociumaka, and Philip Wellnitz. Faster approximate pattern matching: A unified approach. In Sandy Irani, editor, *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020*, pages 978–989. IEEE, 2020. doi:10.1109/FOCS46700.2020.00095.
- 10 Kuei-Hao Chen, Guan-Shieng Huang, and Richard Chia-Tung Lee. Bit-parallel algorithms for exact circular string matching. *The Computer Journal*, 57(5):731–743, March 2013. doi:10.1093/comjnl/bxt023.
- 11 Maxime Crochemore, Costas S. Iliopoulos, Jakub Radoszewski, Wojciech Rytter, Juliusz Straszłyński, Tomasz Waleń, and Wiktor Zuba. Shortest covers of all cyclic shifts of a string. *Theoretical Computer Science*, 866:70–81, 2021. doi:10.1016/j.tcs.2021.03.011.
- 12 Maxime Crochemore, Costas S. Iliopoulos, Jakub Radoszewski, Wojciech Rytter, Juliusz Straszłyński, Tomasz Waleń, and Wiktor Zuba. Linear-time computation of shortest covers of all rotations of a string. In Hideo Bannai and Jan Holub, editors, *33rd Annual Symposium on Combinatorial Pattern Matching, CPM 2022*, volume 223 of *LIPICs*, pages 22:1–22:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICs.CPM.2022.22.
- 13 Maxime Crochemore, Thierry Lecroq, and Wojciech Rytter. *125 Problems in Text Algorithms: With Solutions*. Cambridge University Press, 2021.
- 14 Kimmo Fredriksson and Szymon Grabowski. Average-optimal string matching. *Journal of Discrete Algorithms*, 7(4):579–594, 2009. doi:10.1016/j.jda.2008.09.001.
- 15 Paweł Gawrychowski. Pattern matching in Lempel-Ziv compressed strings: Fast, simple, and deterministic. In Camil Demetrescu and Magnús M. Halldórsson, editors, *Algorithms - ESA 2011 - 19th Annual European Symposium*, volume 6942 of *Lecture Notes in Computer Science*, pages 421–432. Springer, 2011. doi:10.1007/978-3-642-23719-5\_36.
- 16 Roberto Grossi, Costas S. Iliopoulos, Jesper Jansson, Zara Lim, Wing-Kin Sung, and Wiktor Zuba. Finding the cyclic covers of a string. In Chun-Cheng Lin, Bertrand M. T. Lin, and Giuseppe Liotta, editors, *Algorithms and Computation - 17th International Conference and Workshops, WALCOM 2023*, volume 13973 of *Lecture Notes in Computer Science*, pages 139–150. Springer, 2023. doi:10.1007/978-3-031-27051-2\_13.

- 17 Roberto Grossi, Costas S. Iliopoulos, Robert Mercas, Nadia Pisanti, Solon P. Pissis, Ahmad Retha, and Fatima Vayani. Circular sequence comparison: algorithms and applications. *Algorithms for Molecular Biology*, 11:12, 2016. doi:10.1186/s13015-016-0076-6.
- 18 Godfrey H. Hardy and Edward M. Wright. *An Introduction to the Theory of Numbers*. Oxford University Press, 1960.
- 19 Costas S. Iliopoulos, Solon P. Pissis, and M. Sohel Rahman. Searching and indexing circular patterns. In Mourad Elloumi, editor, *Algorithms for Next-Generation Sequencing Data, Techniques, Approaches, and Applications*, pages 77–90. Springer, 2017. doi:10.1007/978-3-319-59826-0\_3.
- 20 Tomasz Kociumaka. *Efficient Data Structures for Internal Queries in Texts*. PhD thesis, University of Warsaw, October 2018. URL: <https://www.mimuw.edu.pl/~kociumaka/files/phd.pdf>.
- 21 Tomasz Kociumaka, Marcin Kubica, Jakub Radoszewski, Wojciech Rytter, and Tomasz Waleń. A linear-time algorithm for seeds computation. *ACM Transactions on Algorithms*, 16(2):27:1–27:23, 2020. doi:10.1145/3386369.
- 22 Tomasz Kociumaka, Jakub Radoszewski, Wojciech Rytter, and Tomasz Waleń. Internal pattern matching queries in a text and applications. In Piotr Indyk, editor, *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015*, pages 532–551. SIAM, 2015. doi:10.1137/1.9781611973730.36.
- 23 M. Lothaire. *Applied Combinatorics on Words*. Cambridge University Press, 2005. URL: [http://www.cambridge.org/gb/knowledge/isbn/item1172552/?site\\_locale=en\\_GB](http://www.cambridge.org/gb/knowledge/isbn/item1172552/?site_locale=en_GB).
- 24 Andrés Marzal, Ramón Mollineda, Guillermo Penis, and Enrique Vidal. Cyclic string matching: Efficient exact and approximate algorithms. In Dechang Chen and Xiuzhen Cheng, editors, *Pattern Recognition and String Matching*, pages 477–497. Springer US, Boston, MA, 2002. doi:10.1007/978-1-4613-0231-5\_19.
- 25 Dennis W. G. Moore and William F. Smyth. An optimal algorithm to compute all the covers of a string. *Information Processing Letters*, 50(5):239–246, 1994. doi:10.1016/0020-0190(94)00045-X.
- 26 Dennis W. G. Moore and William F. Smyth. A correction to "An optimal algorithm to compute all the covers of a string". *Information Processing Letters*, 54(2):101–103, 1995. doi:10.1016/0020-0190(94)00235-Q.
- 27 Vicente Palazón-González and Andrés Marzal. On the dynamic time warping of cyclic sequences for shape retrieval. *Image Vision Computing*, 30(12):978–990, 2012. doi:10.1016/j.imavis.2012.08.012.
- 28 Vicente Palazón-González and Andrés Marzal. Speeding up the cyclic edit distance using LAESA with early abandon. *Pattern Recognition Letters*, 62:1–7, 2015. doi:10.1016/j.patrec.2015.04.013.
- 29 Vicente Palazón-González, Andrés Marzal, and Juan Miguel Vilar. On hidden Markov models and cyclic strings for shape recognition. *Pattern Recognition*, 47(7):2490–2504, 2014. doi:10.1016/j.patcog.2014.01.018.
- 30 Michael J. Tisza, Diana V. Pastrana, Nicole L. Welch, Brittany Stewart, Alberto Peretti, Gabriel J. Starrett, Yuk-Ying S. Pang, Siddharth R. Krishnamurthy, Patricia A. Pesavento, David H. McDermott, et al. Discovery of several thousand highly diverse circular DNA viruses. *eLife*, 9:e51971, 2020. doi:10.7554/eLife.51971.
- 31 Edward K. Wagner, Martinez J. Hewlett, David C. Bloom, and David Camerini. *Basic Virology*, volume 3. Blackwell Science Malden, MA, 1999.