









Finding the Cyclic Covers of a String

Roberto Grossi¹ , Costas S. Iliopoulos² , Jesper Jansson³ , Zara Lim^{2(✉)} ,
Wing-Kin Sung⁴ , and Wiktor Zuba⁵ 

¹ Dipartimento di Informatica, Università di Pisa, Pisa, Italy
`roberto.grossi@unipi.it`

² Department of Informatics, King's College London, London, UK
{`costas.ilopoulos,zara.lim`}@kcl.ac.uk

³ Graduate School of Informatics, Kyoto University, Kyoto, Japan
`jj@i.kyoto-u.ac.jp`

⁴ Department of Computer Science, National University of Singapore,
Singapore, Singapore
`ksung@comp.nus.edu.sg`

⁵ Centrum Wiskunde & Informatica, Amsterdam, The Netherlands
`wiktor.zuba@cwi.nl`

Abstract. We introduce the concept of cyclic covers, which generalizes the classical notion of covers in strings. Given any nonempty string X of length n , a factor W of X is called a cyclic cover if every position of X belongs to an occurrence of a *cyclic shift* of W . Two cyclic covers are distinct if one is not a cyclic shift of the other. The *cyclic cover problem* requires finding all distinct cyclic covers of X . We present an algorithm that solves the cyclic cover problem in $\mathcal{O}(n \log n)$ time. This is based on finding a well-structured set of standard occurrences of a constant number of factors of a cyclic cover candidate W , computing the regions of X covered by cyclic shifts of W , extending those factors, and taking the union of the results.

Keywords: String · Cyclic string · Cover · Periodicity · Regularities

1 Introduction

String periodicities and repetitions have been thoroughly studied in many fields such as string combinatorics, pattern matching and automata theory [25, 26] which can be linked to its importance across various applications, in addition to its theoretical aspects. Detection algorithms and data structures for repeated patterns and regularities span across several fields of computer science [13, 18], for example computational biology, pattern matching, data compression, and randomness testing.

Covers of strings have also been extensively studied in similar fields of combinatorics. The concept originates from quasiperiodicity, a generalization of periodicity which also allows those identical strings to overlap [5]. A factor W of a nonempty string X is called a cover if every position of X belongs to some occurrence of W in X . Furthermore, a cover W must also be a border (i.e. appearing

as both a prefix and a suffix) of the string X . Moore and Smyth [30] developed a linear-time algorithm which computes all covers of a string. Apostolico et al. [6] developed a linear-time algorithm for finding the shortest cover, which Breslauer developed into an on-line algorithm [8]. Li and Smyth [24] produced an on-line algorithm for the all-covers problem. Related string factorization problems include antiperiods [2] and anticovers [1], in addition to approximate [3] and partial [22] covers and seeds [21]. Other combinatorial covering problems consider applications to graphs [11,31].

Cyclic strings have been commonly studied throughout various computer science and mathematical fields, mostly occurring in the field of combinatorics. A cyclic string is a string that does not have an initial or terminal position; instead, the two ends of the string are joined together, and the string can be viewed as a necklace of letters. A cyclic string of length n can be also viewed as a traditional linear string, which has the left- and right-most letters wrapped around and stuck together. Under this notion, the same cyclic string can be seen as n linear strings, which would all be considered equivalent. One of the earliest studies of cyclic strings occurs in Booth’s linear time algorithm [7] for computing the lexicographically smallest cyclic factor of a string. Other closely related works reference terms such as ‘Lyndon factorization’ and ‘canonization’ [4, 10, 14, 16, 27, 28, 33]. Some recent advances on cyclic strings can be found in [12]. Aside from combinatorics, cyclic strings have applications within Computational Biology, such as detecting DNA viruses with circular structures [34, 35].

We introduce the concept of a *cyclic cover* of a nonempty string X of length $n = |X|$, which generalizes the notion of a cover under cyclic shifts. A factor W of X is called a cyclic cover if every position of X belongs to an occurrence of a *cyclic shift* of W . Figure 1 displays an example where X has a cyclic cover where factors have length $\ell = 3, 4, 7, 10, 13, 16$ (cyclic occurrences of the shortest two are shown on the figure).

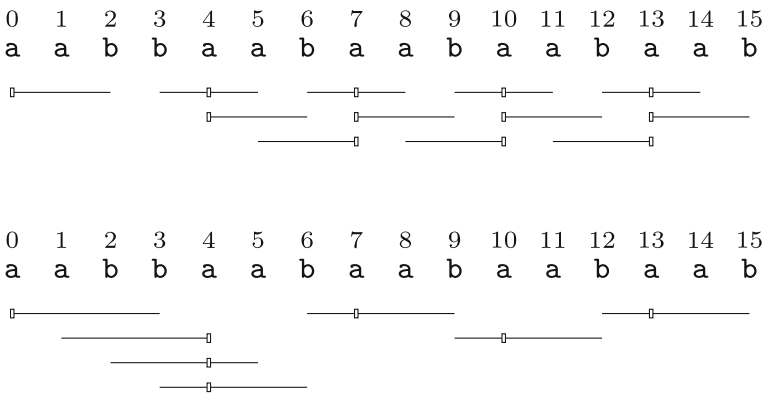


Fig. 1. String $X = \text{aabbaabaabaabaab}$ has a cyclic cover of length 3, as $X[0..2]$, $X[3..5]$, $X[4..6]$, $X[5..7]$, $X[6..8]$, $X[7..9]$, $X[8..10]$, $X[9..11]$, $X[10..12]$, $X[11..13]$, $X[12..14]$, $X[13..15]$ are all cyclic shifts of the same factor, and cover all positions of X . Similarly X has a cyclic cover of length 4.

Looking at the example, we observe that if two distinct factors W and Z , of the same length ℓ , are cyclic covers of X , then W must be a cyclic shift of Z , and vice versa. For this, we say that two cyclic covers are distinct if one is *not* the cyclic shift of the other (in other words, they must have different length as factors of X). Moreover, if a cyclic cover of length ℓ exists, then the prefix of X of length ℓ is also a cyclic cover and, consequently, is the representative of all factors of length ℓ that are cyclic covers of X (as the latter ones are all cyclic shifts of the prefix). Because of this, it is enough to give ℓ as output.

Our Contribution. We introduce the following *cyclic cover problem*: given an input string X of length n , find all the distinct cyclic covers of X , namely, the prefixes of X that are cyclic covers (actually, their lengths ℓ). Under this definition, we have at most n distinct cyclic covers whereas there might be $\Theta(n^2)$ distinct factors that are cyclic covers.¹ In the example of Fig. 1, the output of the cyclic cover problem is $\ell = 3, 4, 7, 10, 13, 16$.

We show that for a string of length n , the cyclic cover problem can be solved in $\mathcal{O}(n \log n)$ time. We assume, that the input string is over a polynomially bounded integer alphabet, and the word RAM model of computation with word size $\mathcal{O}(\log n)$ (both restrictions follow from the restrictions of cited and used data structures).

The rest of the paper is organized as follows. In Sect. 2, we present the preliminary concepts. Section 3 shows our findings for cyclic covers. Finally, we present concluding remarks in Sect. 4.

2 Preliminaries

2.1 Basic Definitions

A *string* X of length $n = |X|$ is a sequence of n letters over an integer alphabet $\Sigma = \{0, \dots, n^{\mathcal{O}(1)}\}$. The letter at position i , for $0 \leq i < n$, is denoted as $X[i]$. A positive integer $p < n$ is called a *period* of X if $X[i] = X[i + p]$ for all $i = 0, \dots, n - p - 1$. By $X[i..j]$, we denote a *factor* of X equal to $X[i] \cdots X[j]$, whereby if $i > j$, then it is the empty string. The factor $X[i..j]$ is a *prefix* of X if $i = 0$, and a *suffix* of X if $j = n - 1$. If $X[0..b-1] = X[n-b..n-1]$, the factor $X[0..b-1]$ is called a *border* of X . A factor W is *periodic* if its smallest period is at most $|W|/2$, and W is *highly-periodic* if its smallest period is at most $|W|/4$. An important property used throughout the paper is Fine and Wilf's periodicity lemma.

Lemma 1 ([15]). *If p, q are periods of a string X of length $|X| \geq p + q - \gcd(p, q)$, then $\gcd(p, q)$ is also a period of X .*

A factor U is a *cyclic shift* of a factor W if $W = AB$ and $U = BA$ for some strings A and B . In that case, we also say that U is a d -cyclic shift of W where

¹ For example, for $X = a^k b a^k$ and $k > 1$, all factors $a^i b a^j$ are cyclic covers of X , for $i, j \geq 0$ such that $i + j \geq k$. They are represented by the prefixes of length $i + j + 1$.

$d = |A|$. (Clearly, $d = 0$ implies that $U = W$ and so there is no cyclic shift.) A factor W is called a *cyclic cover* of X if, for every position i ($0 \leq i < n$), there exists a factor $X[l \dots l + |W| - 1]$ that is a cyclic shift of W and contains position i (i.e. $0 \leq l \leq i \leq l + |W| - 1 < n$). Two cyclic covers are distinct if they are not cyclic shifts of one another. As observed in the introduction, the distinct cyclic covers are represented by (the lengths of) the prefixes of X .

We denote by $\text{lcp}(X[i \dots j], X[k \dots l])$ the length of the longest common prefix of factors $X[i \dots j]$ and $X[k \dots l]$. Also, we denote by $\text{lcp}^r(X[i \dots j], X[k \dots l])$ the length of the longest common suffix of $X[i \dots j]$ and $X[k \dots l]$. Both lcp and lcp^r can be computed in $\mathcal{O}(1)$ time after an $\mathcal{O}(n)$ -time preprocessing of X [19].

2.2 The IPM Data Structure

A useful data structure called the *Internal Pattern Matching* (IPM) data structure was introduced in [23]. The following three lemmas summarize some of its properties. Let us denote by $\text{occ}(W, Z)$ the (possibly empty) list of positions j such that $W = Z[j \dots j + |W| - 1]$.

Lemma 2 ([20, 23]). *Given a string X of length n , the IPM data structure of X after $\mathcal{O}(n)$ time and space construction computes $\text{occ}(A, B)$ for any factors A and B of X where $|A| \leq |B| \leq 2|A|$, in $\mathcal{O}(1)$ time. Furthermore, the list of positions is presented as an arithmetic progression.*

Lemma 3 ([20, 23]). *Given a string X of length n , the IPM data structure of X after $\mathcal{O}(n)$ time and space construction determines if A is a cyclic shift of B in $\mathcal{O}(1)$ time, for any two factors A and B of X .*

Lemma 4 ([23]). *Given a string X of length n , the 2-Period data structure of X after $\mathcal{O}(n)$ time and space construction determines if A is periodic and if that is the case computes its shortest period in $\mathcal{O}(1)$ time for any factor A of X .*

In [23] the structures of Lemmas 2 and 3 are constructed in $\mathcal{O}(n)$ expected time. These constructions were made worst-case in [20]. The structure of Lemma 4 was constructed in $\mathcal{O}(n)$ worst-case time already in [23].

3 Cyclic Covers

Consider a string $X[0 \dots n - 1]$ and its length- ℓ factor $W[0 \dots \ell - 1]$. A straightforward approach to verify if X is cyclically covered by W leads to a quadratic algorithm, as follows: we apply Lemma 3 to test whether $X[i \dots i + \ell - 1]$ is a cyclic shift of $X[0 \dots \ell - 1]$, for all $i = 0, 1, \dots, n - \ell$. If the cyclic shifts of $X[0 \dots \ell - 1]$ cover all positions of X , we report $X[0 \dots \ell - 1]$ as a cyclic cover of $X[0 \dots n - 1]$. Such verification takes $\mathcal{O}(n - \ell)$ time. The cyclic cover problem can be solved by verifying all $\ell \in \{1, \dots, n - 1\}$, which takes $\mathcal{O}(n^2)$ time in total.

Below, we show that this problem can be solved in $\mathcal{O}(n \log n)$ time. Before we detail the algorithm, we first outline 3 techniques:

1. Section 3.1 gives a function $FindFixedCover(W, X, i, j)$ that verifies if X is cyclically covered by W with the constraint that $W[i]$ aligns to $X[j]$.
2. Based on the function $FindFixedCover(W, X, i, j)$, Sect. 3.2 gives an $\mathcal{O}(n/\ell)$ -time algorithm that finds regions in X covered by W when W is highly-periodic.
3. Based on the function $FindFixedCover(W, X, i, j)$, Sect. 3.3 gives an $\mathcal{O}(n/\ell)$ -time algorithm that finds regions in X covered by W when W is not highly-periodic.

3.1 Find Regions in X Covered by Cyclic Shifts of W with the Constraint that $W[i]$ Aligns to $X[j]$

Consider a string $X[0..n-1]$ and its length- ℓ factor $W[0..\ell-1]$. For any $j' \in [j-\ell+1..j]$, a length- ℓ factor $X[j'..j'+\ell-1]$ of X is called a cyclic shift of $W[0..\ell-1]$ with $W[i]$ aligned to $X[j]$ if the i -cyclic shift of W equals the $(j-j')$ -cyclic shift of $X[j'..j'+\ell-1]$. The lemma below computes the region $X[\alpha..\beta]$ in X that is cyclically covered by W with the constraint that $W[i]$ aligns to $X[j]$.

Lemma 5. *Consider a string $X[0..n-1]$ and its length- ℓ factor $W[0..\ell-1]$. Let $\ell_1 = lcp(W[i..\ell-1]W[0..i-1], X[j..n-1])$ and $\ell_2 = lcp^r(W[i+1..\ell-1]W[0..i], X[0..j]) - 1$. If $\ell_1 + \ell_2 \geq \ell$, then $X[j-\ell_2..j+\ell_1-1]$ is cyclically covered by W with the constraint that $W[i]$ aligns to $X[j]$; otherwise, such a cyclic cover does not exist.*

Proof. Let $U = W[i..\ell-1]W[0..i-1]$. Observe that $X[j-\ell_2..j+\ell_1-1] = U^2[\ell-\ell_2..\ell+\ell_1-1]$ from the definitions of ℓ_1 and ℓ_2 . Every factor of length ℓ of this string is a cyclic shift of U , hence also of W , thus it is cyclically covered by W , with the constraint that $U[0]$ aligns to $X[j]$ (hence $W[i]$ aligns to $X[j]$).

If $\ell_1 + \ell_2 < \ell$, then X does not contain any cyclic shift of U , with $U[0]$ aligned to $X[j]$ since $U[\ell-\ell_2-1] \neq X[j-\ell_2-1]$ and $U[\ell_1] \neq X[j+\ell_1]$, and any such cyclic shift (as a factor of X) would contain one of those two positions (those positions are less than ℓ positions apart, and position j is in between them). \square

For example, given the word $X = babbababb$, the factor $W = bbab$ and the constraint that $X[3]$ aligns with $W[1]$, $X[1..6]$ is cyclically covered by W (see Fig. 2).

$W = bbab$ and $X = abbbab$ where $W[1]$ aligns with $X[3]$. Thus W cyclically covers the region $X[1..6]$.

Based on the above lemma, we denote $FindFixedCover(W, X, i, j)$ as the function that returns the region $X[\alpha..\beta]$ that is cyclically covered by W with the constraint that $W[i]$ aligns to $X[j]$. If no such cyclic cover exists, the function returns an empty region.

Lemma 6. *After linear time preprocessing, we can compute $FindFixedCover(W, X, i, j)$ for any factor W of X in $\mathcal{O}(1)$ time.*

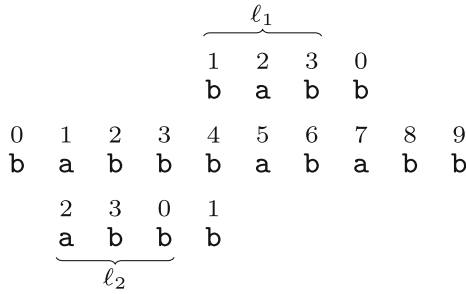


Fig. 2. Given $X = babbababb$, $W = bbab$ and the constraint that $X[3]$ aligns with $W[1]$, the factor $X[1..6]$ is cyclically covered by W . The lengths ℓ_1 and ℓ_2 denote $lcp(W[1..3]W[0], X[4..9])$ and $lcp^r(W[2..3]W[0..1], X[0..4]) - 1$, respectively.

Proof. We first build the lcp data structures for X and for its reverse in linear time and then use them to compute ℓ_1 and ℓ_2 . Even though $W[i..i-1]W[0..i-1]$ does not need to occur in X in such a case, we simply compute ℓ_1 in two steps. If $lcp(W[i..i-1], X[j..n-1]) < i - j$, then it represents the sought value. Otherwise $\ell_1 = (i - j) + lcp(W[0..i-1], X[j+i..n-1])$. Note that ℓ_2 is computed analogously. □

3.2 Finding Regions in X that are Cyclically Covered by a Highly-Periodic Factor W

Lemma 8 describes how to find regions that are cyclically covered by $W[0..i-1]$ if W is of period q where $q \leq \ell/4$. To show it, we make use of Lemma 7 from [29] (see also [9, 17]) to represent occurrences in a convenient way. Below, we let (j_1, q, m) denote the arithmetic progression j_1, j_2, \dots, j_m with $j_{s+1} = j_s + q$, where $1 \leq s < m$.

Lemma 7 ([29], Lemma 3.1). *Suppose the minimum period of $W[0..i-1]$ is q . For a length- 2ℓ factor Y , $occ(W, Y)$ equals a single arithmetic progression (j_1, q', m') . If $m' \geq 3$, then $q' = q$.*

Lemma 8. *Suppose the smallest period of $W[0..i-1]$ is $q \leq \ell/4$. We can find which parts of $X[i..i+\ell-1]$ are cyclically covered by W in $\mathcal{O}(1)$ time.*

Proof. Any cyclic shift of W that covers any position of $X[i..i+\ell-1]$ must be fully contained inside $X[i-\ell..i+2\ell-1]$, hence we are going to restrict our search to that region.

Let $Y = W[0.. \lfloor \ell/2q \rfloor q - 1]$, which is $W[0..q-1]^{\lfloor \ell/2q \rfloor}$. Note that $\ell/3 < |Y| \leq \ell/2$, and also $|Y| \geq 2q$, hence q is its smallest period (a smaller period would imply a smaller period of X by Lemma 1). Any cyclic shift of W must contain Y as a factor.

We first find the occurrences of Y in $X[i-\ell..i+2\ell-1]$. By Lemma 2, these occurrences can be found in $\mathcal{O}(1)$ time by computing $occ(Y, X[i'..i'+2|Y|])$ for

$i' \in \{i - \ell + h|Y| \mid h = 0, 1, 2, \dots \lfloor 3\ell/|Y| \rfloor\}$. Since $3\ell/|Y| < 9$ we have at most 9 arithmetic progressions with period q (by Lemma 7) plus up to 18 standalone occurrences.

For each standalone occurrence starting at position j we can simply run $FindFixedCover(W, X, 0, j)$ separately. Processing of the arithmetic progressions is a little more complex however.

To see how to do it efficiently, we first prove a crucial claim. For an arithmetic progression (j_1, q, m) and $1 \leq s \leq m$, let $X[\alpha_s \dots \beta_s] = FindFixedCover(W, X, 0, j_s)$. We claim that the following inequalities hold: $\beta_s \leq \beta_{s+1}$ and $\alpha_s \leq \alpha_{s+1}$. The former inequality $\beta_s = j_s + lcp(W, X[j_s \dots n-1]) - 1 \leq j_{s+1} + lcp(W[0 \dots q-1]W, X[j_s \dots n-1]) - 1 = j_s + q + lcp(W, X[j_{s+1} \dots n-1]) - 1 = \beta_{s+1}$ is simple to see as W is a prefix of $W[0 \dots q-1]W$. For the latter inequality $\alpha_s \leq \alpha_{s+1}$, notice that if $|W|$ is a multiple of q , then we can simply apply a proof symmetric to the one for β 's. Otherwise $\alpha_{s+1} < \alpha_s \leq j_s$ for $s \geq 1$ would imply a non-trivial border of W of length q , which in turn would imply that $W - |q|$ is a period of W . By Lemma 1, we have $gcd(q, |W| - q) < q$, which is a contradiction. This completes the proof of the claim.

Due to this claim, the region obtained for this sequence is $X[\alpha_1 \dots \beta_m]$, and only two calls of $FindFixedCover$ are needed. □

In conclusion, as factors $X[k\ell \dots (k+1)\ell - 1]$ for $k \in [0, \lfloor \frac{n}{\ell} \rfloor - 1]$ and $X[n - \ell \dots n - 1]$ contain all positions of X , we have the following corollary.

Corollary 1. *After an $\mathcal{O}(n)$ time preprocessing of the string $X[0 \dots n - 1]$, for any highly-periodic factor $W[0 \dots \ell - 1]$, we can compute the regions in X which are cyclically covered by W in $\mathcal{O}(n/\ell)$ time.*

3.3 Finding Regions in X that Are Cyclically Covered by a Non-highly-periodic Factor W

The lemma below states that factors that are not highly-periodic do not occur frequently in X , and follows directly from the definition of a period.

Lemma 9. *Consider a string $X[0 \dots n - 1]$ and a non-highly-periodic factor $W[0 \dots \ell - 1]$. Any two occurrences of W in X are at distance at least $\ell/4$.*

Let W' be some factor of W . If a cyclic shift of W contains W' , we call it a W' -containing cyclic shift of W .

Consider $W[0 \dots \ell - 1] = W_l W_r$ where $|W_l| = \lfloor \ell/2 \rfloor$. The following lemma gives a way to find all regions in X covered by cyclic shifts of W .

Lemma 10. *Consider $W[0 \dots \ell - 1] = W_l W_r$ where $|W_l| = \lfloor \ell/2 \rfloor$. For a string X , let A be the set of all regions in X covered by W_l -containing cyclic shifts of $W_l W_r$, and let B be the set of all regions in X covered by W_r -containing cyclic shifts of $W_r W_l$. Then $A \cup B$ forms the set of all regions in X that are cyclically covered by W .*

Proof. Observe that every cyclic shift of W must contain either W_l or W_r . Hence, the lemma follows. \square

Below we focus on describing an algorithm that finds all regions in X covered by W_l -containing cyclic shifts of W_lW_r . All regions in X covered by W_r -containing cyclic shifts of W_rW_l can be found by an analogous algorithm.

To find all regions in X covered by W_l -containing cyclic shifts of W_lW_r , we consider two cases: W_l is highly-periodic or not.

If W_l is not highly-periodic, then it has $\mathcal{O}(n/\ell)$ occurrences in $X[0..n-1]$ (Lemma 9). Thus we can find all these occurrences in $\mathcal{O}(n/\ell)$ time given the IPM data structure (Lemma 3). Then, using *FindFixedCover()*, the regions in X covered by these W_l -containing cyclic shifts of W can be found using $\mathcal{O}(n/\ell)$ time.

For a highly periodic W_l , let $q_l \leq \ell/8$ denote its shortest period, and let d_l denote the longest prefix of W which is q_l -periodic. Let us also denote $W_{l'} = W[0..d_l-1]$ and $W_{r'} = W[d_l..\ell-1]$. Notice that if $W_{r'}W_{l'}$ is highly-periodic we can simply reduce our problem to the case with a highly-periodic W as any cyclic shift of W is also a cyclic shift of $W_{r'}W_{l'}$. Notice, also, that a W_l -containing cyclic shift of W (d -cyclic shift of W for $d = 0$ or $d \geq |W_l|$) is always a $W_{l'}W[d_l]$ -containing factor of W (for $d = 0$ or $d > d_l$) or a $W_{r'}W_l$ -containing factor of W (for $|W_l| \leq d \leq d_l$).

Now it is enough to show that, for a highly-periodic W_l when W and $W_{r'}W_{l'}$ are not highly-periodic, $W_{l'}W[d_l]$ and $W_{r'}W_l$ are not highly-periodic as well.

Lemma 11. *$W_{l'}W[d_l]$ is non-periodic (hence also non-highly-periodic).*

Proof. By contradiction, suppose that $W[0..d_l] = W_{l'}W[d_l]$ has period $q' \leq (d_l + 1)/2$. This means that $W_{l'}$ has both periods q_l and q' . Since $q_l + q' \leq \ell/8 + (d_l + 1)/2 \leq d_l$, we have that $\gcd(q_l, q')$ is also a period of $W_{l'}$ by Lemma 1.

We observe that q' cannot be a multiple of q_l as in this case $W[d_l] = W[d_l - q'] = W[d_l - q]$, which contradicts the definition of d_l . Hence we get $\gcd(q_l, q') < q_l$, which in turn contradicts the fact that q_l is the shortest period of $W_{l'}$. \square

Lemma 12. *$W_{r'}W_l$ is not highly-periodic.*

Proof. Suppose, on the contrary, that $W_{r'}W_l$ has period $q' \leq |W_{r'}W_l|/4 \leq \ell/4$. This means, that W_l has both periods q_l and q' . Since $q_l + q' \leq \ell/2$ by Lemma 1 $\gcd(q_l, q')$ is also a period of W_l .

If q' is a multiple of q_l , then $W_{r'}W_{l'}$ is also $q' \leq \ell/4$ periodic contrary to the assumptions, otherwise $\gcd(q_l, q') < q_l$ which contradicts that q_l is the shortest period of W_l . \square

Now, we are ready to describe a function *FindCyclicCover*(W_lW_r, X) that returns all regions in X that are covered by W_l -containing cyclic shifts of W . This function is described in Algorithm 1.

Algorithm 1. *FindCyclicCover*(W_l, W_r, X)**Output:** Regions in X covered by W_l -containing cyclic shifts of W

-
- 1: If $W = W_l W_r$ or $W_r W_l$ is of period $\leq \ell/4$, we apply Corollary 1 to find the regions of X covered by W using $\mathcal{O}(n/\ell)$ time and return the answer.
 - 2: $Ans = \emptyset$
 - 3: **if** W_l is not highly-periodic **then**
 - 4: Find j_1, \dots, j_m such that $X[j_s \dots j_s + |W_l| - 1] = W_l$ using $\mathcal{O}(n/\ell)$ time.
 - 5: For each j_s , $Ans = Ans \cup FindFixedCover(W, X, 0, j_s)$
 - 6: **else**
 - 7: Find j_1, \dots, j_m such that $X[j_s \dots j_s + d_l] = W_l W[d_l]$ using $\mathcal{O}(n/\ell)$ time.
 - 8: For each j_s , $Ans = Ans \cup FindFixedCover(W, X, 0, j_s)$
 - 9: Find j_1, \dots, j_m such that $X[j_s \dots j_s + |W_r W_l| - 1] = W_r W_l$ using $\mathcal{O}(n/\ell)$ time.
 - 10: For each j_s , $Ans = Ans \cup FindFixedCover(W, X, d_l, j_s)$
 - 11: **end if**
 - 12: Return Ans
-

Lemma 13 summarizes the time complexity of *FindCyclicCover*(W_l, W_r, X).

Lemma 13. *Given the lcp, IPM and 2-Period data structures of X , we can compute *FindCyclicCover*(W_l, W_r, X) (and *FindCyclicCover*(W_r, W_l, X)) in $\mathcal{O}(n/\ell)$ time.*

Proof. Let us first assume that we know an occurrence in X of any given string. To check whether W and W_l are (highly-)periodic, it is enough to perform the 2-Period queries (Lemma 4). Later, with the use of a single *lcp* query (*lcp*($X, X[q_l \dots n-1]$) in this case), one can compute d_l . $W_r W_l$ can only be highly periodic if W_l is periodic with the same period, hence a check of whether it is highly periodic only requires a comparison between parts of W_l and W_r which takes $\mathcal{O}(1)$ time in total. After determining which method to use, the algorithm performs $\mathcal{O}(n/\ell)$ *FindFixedCover*() queries, which results in a $\mathcal{O}(n/\ell)$ total time complexity.

In general, we do not know the occurrences of some of the strings (for example $W_r W_l$), or even if they occur in X at all. To address this issue and be able to use the internal data structures we make some adjustments.

For the cyclic shifts of W , namely, $W_r W_l$, $W_r W_l$ and its counterpart used by *FindCyclicCover*(W_r, W_l, X), we only need to check whether they are highly-periodic and employ the *lcp* (or *lcp^r*) with another string. To address the first point, it is sufficient to check whether their longest factor which appears in W is periodic, and whether the period can be extended to the whole string (with *lcp* queries). This factor must be of length at least $\ell/2$; hence, it must be periodic if the whole string is highly-periodic. Its shortest period is the only candidate for the shortest period (of length at most $\ell/4$) of the whole string. As for the second point, *lcp*, this is only used by Lemma 6, where this problem has already been solved.

Another string which does not need to appear in X is $W_r W_l$ (symmetrically $(W_r W_l)[0 \dots d_r]$ used by *FindCyclicCover*(W_r, W_l, X)). We make use of

this string only if W_l is highly periodic. Using the lcp^r query, we can find how far this period extends to the left in $W_r W_l$. Now, instead of looking for the whole $W_r W_l$ in the parts of X , we simply look for W_l . If a whole arithmetic sequence (j_1, q_l, m) of occurrences is found, then we know that only one of those occurrences can be extended to the whole $W_r W_l$ (with j_{k+1} , where k is equal to the number of periods of W_l at the end of W_r). This way we can process the whole X in $\mathcal{O}(n/\ell)$ time. \square

Theorem 1 (Cyclic cover problem). *Given a string X of length n , over an integer alphabet, we can find all integers $\ell > 0$ such that the prefix $W = X[0.. \ell - 1]$ is a cyclic cover of X , in $\mathcal{O}(n \log n)$ total time.*

Proof. In the preprocessing step we construct the Internal Data Structure answering lcp, IPM and 2-Period queries in $\mathcal{O}(n)$ time (Lemmas 2 and 4). For any fixed ℓ , let $W_l = X[0.. \lfloor \ell/2 \rfloor - 1]$ and $W_r = X[\lfloor \ell/2 \rfloor .. \ell - 1]$. We can check if $W = X[0.. \ell - 1]$ is a cyclic cover of $X[0.. n - 1]$ by applying $FindCyclicCover(W_l, W_r, X)$ and $FindCyclicCover(W_r, W_l, X)$. Lemma 13 shows that these two functions run in $\mathcal{O}(\frac{n}{\ell})$ time. The total time to test $\ell = 1, \dots, n$ is upper bounded by $\mathcal{O}(\sum_{\ell=1}^n \frac{n}{\ell}) = \mathcal{O}(n \log n)$. \square

4 Concluding Remarks

In this paper we showed that all distinct cyclic covers can be found in $\mathcal{O}(n \log n)$ time. The techniques introduced in our solution can also give (much simpler) algorithms for two other related problems.

The first one is to find all cyclic borders of X , namely, all values of ℓ such that prefix $X[0.. \ell - 1]$ is a cyclic shift of suffix $X[n - \ell .. n - 1]$. It can be solved in $\mathcal{O}(n)$ time by simply using Lemma 3 n times.

The second problem is to find all the cyclic factorizations, which are a special case of the cyclic covers: X is partitioned into factors of length ℓ , for all feasible ℓ , so that each resulting factor is a cyclic shift of the others. We obtain an $\mathcal{O}(n \log \log n)$ time algorithm by using Lemma 3 $\mathcal{O}(\frac{n}{\ell})$ times for every length ℓ that divides n (denoted as $\ell|n$). The complexity follows from the bound $\sum_{\ell|n} \frac{n}{\ell} = \mathcal{O}(n \log \log n)$ given in [32, Thm.2].

References

1. Alzamel, M., et al.: Finding the anticover of a string. In: 31st Annual Symposium on Combinatorial Pattern Matching (CPM 2020), vol. 161 (2020)
2. Alzamel, M., et al.: Online algorithms on antipowers and antiperiods. In: Brisaboa, N.R., Puglisi, S.J. (eds.) SPIRE 2019. LNCS, vol. 11811, pp. 175–188. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-32686-9_13
3. Amir, A., Levy, A., Lubin, R., Porat, E.: Approximate cover of strings. Theor. Comput. Sci. **793**, 59–69 (2019). <https://doi.org/10.1016/j.tcs.2019.05.020>
4. Apostolico, A., Crochemore, M.: Fast parallel Lyndon factorization with applications. Math. Syst. Theor. **28**(2), 89–108 (1995). <https://doi.org/10.1007/BF01191471>

5. Apostolico, A., Ehrenfeucht, A.: Efficient detection of quasiperiodicities in strings. *Theor. Comput. Sci.* **119**(2), 247–265 (1993)
6. Apostolico, A., Farach, M., Iliopoulos, C.S.: Optimal superprimitivity testing for strings. *Inf. Process. Lett.* **39**(1), 17–20 (1991)
7. Booth, K.S.: Lexicographically least circular substrings. *Inf. Process. Lett.* **10**(4–5), 240–242 (1980)
8. Breslauer, D.: An on-line string superprimitivity test. *Inf. Process. Lett.* **44**(6), 345–347 (1992)
9. Breslauer, D., Galil, Z.: Real-time streaming string-matching. *ACM Trans. Algorithms* **10**(4), 1–12 (2014). <https://doi.org/10.1145/2635814>
10. Černý, A.: Lyndon factorization of generalized words of Thue. *Discrete Math. Theor. Comput. Sci.* **5**, 17–46 (2002)
11. Conte, A., Grossi, R., Marino, A.: Large-scale clique cover of real-world networks. *Inf. Comput.* **270**, 104464 (2020)
12. Crochemore, M., et al.: Shortest covers of all cyclic shifts of a string. *Theor. Comput. Sci.* **866**, 70–81 (2021)
13. Crochemore, M., Rytter, W.: *Jewels of Stringology: Text Algorithms*. World Scientific, Singapore (2002)
14. Duval, J.P.: Factorizing words over an ordered alphabet. *J. Algorithms* **4**(4), 363–381 (1983)
15. Fine, N.J., Wilf, H.S.: Uniqueness theorems for periodic functions. *Proc. Am. Math. Soc.* **16**(1), 109–114 (1965). <https://doi.org/10.2307/2034009>
16. Fredricksen, H., Maiorana, J.: Necklaces of beads in k colors and k -ary de Bruijn sequences. *Discret. Math.* **23**(3), 207–210 (1978)
17. Galil, Z.: Optimal parallel algorithms for string matching. *Inf. Control.* **67**(1–3), 144–157 (1985). [https://doi.org/10.1016/S0019-9958\(85\)80031-0](https://doi.org/10.1016/S0019-9958(85)80031-0)
18. Gusfield, D.: Algorithms on strings, trees, and sequences: computer science and computational biology. *ACM Sigact News* **28**(4), 41–60 (1997)
19. Kärkkäinen, J., Sanders, P.: Simple linear word suffix array construction. In: Baeten, J.C.M., Lenstra, J.K., Parrow, J., Woeginger, G.J. (eds.) *ICALP 2003*. LNCS, vol. 2719, pp. 943–955. Springer, Heidelberg (2003). https://doi.org/10.1007/3-540-45061-0_73
20. Kociumaka, T.: Efficient data structures for internal queries in texts. Ph.D. thesis, University of Warsaw, October 2018 (2018). <https://www.mimuw.edu.pl/kociumaka/files/phd.pdf>
21. Kociumaka, T., Kubica, M., Radoszewski, J., Rytter, W., Waleń, T.: A linear time algorithm for seeds computation. In: Proceedings of the 23rd Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 1095–1112. SIAM (2012)
22. Kociumaka, T., Pissis, S.P., Radoszewski, J., Rytter, W., Waleń, T.: Fast algorithm for partial covers in words. *Algorithmica* **73**(1), 217–233 (2014). <https://doi.org/10.1007/s00453-014-9915-3>
23. Kociumaka, T., Radoszewski, J., Rytter, W., Waleń, T.: Internal pattern matching queries in a text and applications. In: Proceedings of the 26th Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 532–551. SIAM (2014)
24. Li, Y., Smyth, W.F.: Computing the cover array in linear time. *Algorithmica* **32**(1), 95–106 (2002). <https://doi.org/10.1007/s00453-001-0062-2>
25. Lothaire, M.: *Applied combinatorics on words*. Encyclopedia of Mathematics and its Applications, Cambridge University Press (2005). <https://doi.org/10.1017/CBO9781107341005>
26. Lothaire, M.: *Algebraic Combinatorics on Words*, vol. 90. Cambridge University Press, New York (2002)

27. Melançon, G.: Lyndon factorization of infinite words. In: Puech, C., Reischuk, R. (eds.) STACS 1996. LNCS, vol. 1046, pp. 147–154. Springer, Heidelberg (1996). https://doi.org/10.1007/3-540-60922-9_13
28. Melançon, G.: Lyndon factorization of Sturmian words. *Discret. Math.* **210**(1–3), 137–149 (2000)
29. Miyazaki, M., Shinohara, A., Takeda, M.: An improved pattern matching algorithm for strings in terms of straight-line programs. In: Apostolico, A., Hein, J. (eds.) CPM 1997. LNCS, vol. 1264, pp. 1–11. Springer, Heidelberg (1997). https://doi.org/10.1007/3-540-63220-4_45
30. Moore, D., Smyth, W.F.: Computing the covers of a string in linear time. In: Proceedings of the 5th Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 511–515. SODA’94, Society for Industrial and Applied Mathematics, USA (1994)
31. Norman, R.Z., Rabin, M.O.: An algorithm for a minimum cover of a graph. *Proc. Am. Math. Soc.* **10**(2), 315–319 (1959)
32. Robin, G.: Grandes valeurs de la fonction somme des diviseurs et hypothese de Riemann. *J. Math. Pures Appl.* **63**, 187–213 (1984)
33. Shiloach, Y.: Fast canonization of circular strings. *J. Algorithms* **2**(2), 107–121 (1981)
34. Tisza, M.J., et al.: Discovery of several thousand highly diverse circular DNA viruses. *Elife* **9**, e51971 (2020)
35. Wagner, E.K., Hewlett, M.J., Bloom, D.C., Camerini, D.: *Basic Virology*, vol. 3. Blackwell Science, Malden, MA (1999)