Inf. Process. Lett. 191 (2026) 106594

Contents lists available at ScienceDirect

Information Processing Letters

journal homepage: www.elsevier.com/locate/ipl

Finding the cyclic covers of a string ☆

Roberto Grossi [a], Costas S. Iliopoulos [b], Jesper Jansson [c], Zara Lim [b,*], Wing-Kin Sung [d,e], Wiktor Zuba [f]

[a] Dipartimento di Informatica, Università di Pisa, Italy
[b] Department of Informatics, King's College London, UK
[c] Graduate School of Informatics, Kyoto University, Kyoto, Japan
[d] The Chinese University of Hong Kong, Hong Kong, China
[e] Hong Kong Genome Institute, Hong Kong Science Park, Hong Kong, China
[f] Centrum Wiskunde & Informatica, Amsterdam, the Netherlands

ARTICLE INFO

Keywords:
String
Cyclic string
Cover
Periodicity
Regularities

ABSTRACT

We introduce the concept of cyclic covers, which generalizes the classical notion of covers in strings. Given any string X, a factor W of X is called a *cyclic cover* if each position of X belongs to an occurrence of a cyclic shift of W in X. Two cyclic covers are distinct if one is not a cyclic shift of the other. The *cyclic covers problem* asks for all distinct cyclic covers of an input string X. We present an algorithm that solves the cyclic covers problem in $\mathcal{O}(n \log n)$ time, where n is the length of X. It 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.

1. Introduction

Repetitions and periodicities in strings have been extensively studied across many fields including string combinatorics, pattern matching and automata theory [29,30] due to their theoretical significance and real-world applications. Detection algorithms and data structures for repeated patterns and regularities span across several fields of computer science [14,21], such as computational biology, pattern matching, data compression, and randomness testing.

String covers, a generalization of periodicity, stem from quasiperiodicity [5] and allow occurrences of a repeated factor in a string to overlap. A factor W of a string X is called a cover of X if each position of X belongs to an occurrence of W. See Fig. 1 for an example. By definition, a cover of X must also be a border of X, i.e., it must appear as both a prefix and a suffix of X. Apostolico et al. [6] presented the first linear-time algorithm for finding the shortest cover of a string. Subsequently, Breslauer [9] developed a linear-time on-line algorithm for the problem. Moore and Smyth [34] gave a linear-time algorithm that computes *all* the covers of a string; this result was later extended to a linear-time on-line algorithm by Li and Smyth [28]. Related string factorization problems include antiperiods [2] and anticovers [1], in addition to approximate [3] and partial [26] covers and seeds [25]. Other combinatorial covering problems consider applications to graphs [12,35].

Cyclic strings have been studied throughout various computer science and mathematical fields, in particular 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 [8] for computing the lexicographically smallest cyclic factor of a string. Other closely related works reference terms such as

☆ A preliminary version of this article appeared in *Proceedings of the Seventeenth International Conference and Workshops on Algorithms and Computation* (WALCOM 2023), Lecture Notes in Computer Science, Vol. 13973, pp. 139–150, Springer, 2023.

* Corresponding author.
E-mail addresses: roberto.grossi@unipi.it (R. Grossi), costas.iliopoulos@kcl.ac.uk (C.S. Iliopoulos), jj@i.kyoto-u.ac.jp (J. Jansson), zara.lim@kcl.ac.uk (Z. Lim), kwksung@cuhk.edu.hk (W.-K. Sung), w.zuba@mimuw.edu.pl, wiktor.zuba@cwi.nl (W. Zuba).

https://doi.org/10.1016/j.ipl.2025.106594
Received 3 July 2024; Received in revised form 8 June 2025; Accepted 11 June 2025
Available online 16 June 2025

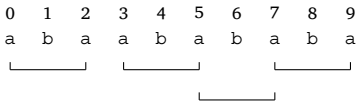

Fig. 1. The string aba is a cover of the string abaabababa.

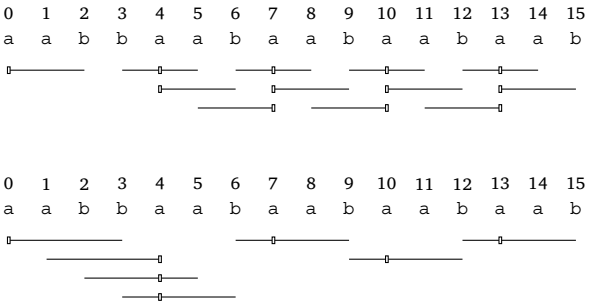

Fig. 2. The string $X = \mathtt{aabbaabaabaabaab}$ has a cyclic cover of length 3 (shown at the top) since $X[0..2]$, $X[3..5]$, $X[4..6]$, $X[5..7]$, $X[6..8]$, $X[7..9]$, $X[8..10]$, $X[9..11]$, $X[10..12]$, $X[11..13]$, $X[12..14]$, $X[13..15]$ are all cyclic shifts of the same factor aab, and cover all positions of X. In addition, X has a cyclic cover of length 4 (shown at the bottom) as well as cyclic covers of lengths 7, 10, 13, and 16 (not shown).

'Lyndon factorization' and 'canonization' [4,11,17,19,31,32,37]. Some recent advances on cyclic strings can be found in [13]. Aside from combinatorics, cyclic strings have applications within bioinformatics [38, 39] or image processing [7,36].

This article introduces the concept of a *cyclic cover* of a string, which generalizes the notion of a cover by incorporating cyclic shifts. A factor W of X is called a cyclic cover if each position of X belongs to an occurrence of a cyclic shift of W in X. Our motivation for cyclic covers originates from viral genomes, such as *Escherichia coli* (*E. coli*), which can form circular sequences [40,38]. The study of cyclic covers in this context generalizes traditional sequence alignment of viral genomes, a crucial tool in bioinformatics for analyzing evolutionary relationships [16].

Fig. 2 shows an example in which X has several different cyclic covers. In the figure, the cyclic occurrences of a cyclic cover of length 3 as well as the cyclic occurrences of one of length 4 are illustrated.

We can immediately note the following:

Lemma 1. *If W is a cyclic cover of length ℓ of a string X then W and the prefix of X of length ℓ are cyclic shifts of each other.*

Proof. By the definition of a cyclic cover, the first position of X belongs to an occurrence of a cyclic shift of W. Since W has length ℓ, the first ℓ positions of X form a cyclic shift of W. □

Two cyclic covers are called distinct if they are not cyclic shifts of one another. By Lemma 1, any two factors W and Z of X of equal length that both are cyclic covers of X have to be cyclic shifts of the same prefix of X, and hence by transitivity, cyclic shifts of each other. For this reason, two cyclic covers of X are distinct if and only if they have different lengths.

Moreover, Lemma 1 implies that whenever a prefix P of X is a cyclic cover, all the cyclic covers of X of that length are cyclic shifts of P. In other words, for any positive integer ℓ, all the length-ℓ cyclic covers of X can be represented by the prefix of X of length ℓ. To obtain the distinct cyclic covers of X, it is therefore sufficient to compute all ℓ for which the length-ℓ prefix of X is a cyclic cover of X.

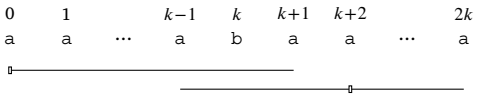

Fig. 3. Let $X = \mathtt{a}^k\mathtt{b}\mathtt{a}^k$ for some integer $k \geq 1$. Every factor of X of the form $\mathtt{a}^i\mathtt{b}\mathtt{a}^j$, where $i, j \geq 0$ are integers such that $i + j \geq k$, is a cyclic cover of X. These $\Theta(k^2)$ cyclic covers are represented by the $\Theta(k)$ prefixes of X of length $k+1, k+2, \ldots, 2k+1$. Indicated in the figure is the cyclic cover given by the prefix of length $k+2$.

1.1. Our contributions

We study the *cyclic covers problem*, which asks for all of the distinct cyclic covers of a string: Given an input string X, output the lengths of all prefixes of X that are cyclic covers of X. As an example, for the input shown in Fig. 2, the output of the cyclic covers problem should be $3, 4, 7, 10, 13, 16$.

The main result in the article is an algorithm that solves the cyclic covers problem in $\mathcal{O}(n \log n)$ time for any string of length n. We assume that the input string is over a general ordered alphabet and that the model of computation is the word RAM model with word size $\Theta(\log n)$; both restrictions follow from the restrictions of cited and used data structures.

The rest of the article is organized as follows. Section 2 provides the formal definitions, reviews some results from the literature, and describes a straightforward quadratic-time algorithm for the cyclic covers problem. Section 3 presents our faster algorithm. Finally, we give some concluding remarks in Section 4.

2. Preliminaries

2.1. Basic definitions

A *string* X of length $n = |X|$ is a sequence of n characters over an integer alphabet $\Sigma = \{0, \ldots, n^{\mathcal{O}(1)}\}$. For every $i \in \{0, 1, \ldots, n-1\}$, the character at position i of the string is denoted by $X[i]$. A positive integer $p < n$ is called a *period of* X if $X[i] = X[i+p]$ for all $i = 0, \ldots, n-p-1$. 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 article is Fine and Wilf's periodicity lemma:

Lemma 2 *([18]). 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.*

Next, a factor U is a *cyclic shift* of a factor W if $W = AB$ and $U = BA$ for some strings A and B. When this condition holds, we say that U is a *d-cyclic shift of W*, where $d = |A|$. A factor W is called a *cyclic cover of X* if, for every position i ($0 \leq i < n$), there exists a factor $X[j..j+|W|-1]$ that is a cyclic shift of W and contains position i, i.e., $0 \leq j \leq i \leq j + |W| - 1 < n$. (For some examples, refer to Figs. 2 and 3.) Two cyclic covers are *distinct* if they are not cyclic shifts of one another. As observed in the introduction, the distinct cyclic covers of X can be represented by the lengths of their corresponding prefixes of X. Consequently, the *cyclic covers problem* is to output the lengths of all prefixes of an input string X that are cyclic covers of X.

Remark. Even though any string of length n has at most n distinct cyclic covers, it could have $\Theta(n^2)$ distinct factors that are cyclic covers, as demonstrated in Fig. 3. Thus, the number of distinct factors that are cyclic covers can be much larger than the number of distinct cyclic covers.

We denote by $lcp(X[i..j], X[k..l])$ the length of the longest common prefix of the factors $X[i..j]$ and $X[k..l]$. Also, we denote by $lcp^r(X[i..j], X[k..l])$ the length of the longest common suffix of $X[i..j]$ and $X[k..l]$. After $\mathcal{O}(n)$ time preprocessing of X, lcp and lcp^r for any two specified factors of X can be computed in $\mathcal{O}(1)$ time [23].

2.2. The IPM data structure

A useful data structure called the *Internal Pattern Matching* (IPM) data structure was introduced in [27]. It efficiently determines if a substring x occurs within another substring y of a given text in constant time, provided that $|y| = O(|x|)$.

The following three lemmas summarize some of its properties. Let us denote by $occ(W, Z)$ the (possibly empty) list of positions j such that $W = Z[j..j + |W| - 1]$.

Lemma 3 *([24,27]). Given a string X of length n, the IPM data structure for X after $\mathcal{O}(n)$ time and space construction computes $occ(A, B)$ for any factors A and B of X where $|A| \le |B| \le 2|A|$, in $\mathcal{O}(1)$ time. Furthermore, the list of positions is presented as an arithmetic progression.*

Lemma 4 *([24,27]). Given a string X of length n, the IPM data structure for 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 5 *([27]). 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 [27], the data structures of Lemmas 3 and 4 were constructed in $\mathcal{O}(n)$ expected time. These constructions were strengthened to $\mathcal{O}(n)$ worst-case time in [24]. It was already shown in [27] how to construct the data structure of Lemma 5 in $\mathcal{O}(n)$ worst-case time.

2.3. A quadratic-time algorithm for the cyclic covers problem

A straightforward approach based on repeated applications of Lemma 4 leads to a quadratic-time algorithm for the cyclic covers problem.

More precisely, first construct the IPM data structure for X in a preprocessing step. Next, for each $\ell \in \{1, 2, \ldots, n-1\}$ independently of each other, check if the length-ℓ prefix of X is a cyclic cover of X as follows: Initialize a variable *covered* to -1 that says which positions of X have been covered so far. Then, for each i from 0 to $n - \ell$, apply Lemma 4 to test whether $X[i..i + \ell - 1]$ is a cyclic shift of $X[0..\ell - 1]$; if the answer is yes then update *covered* to $i + \ell - 1$, but if the answer is no and *covered* $< i$ then go on to the next ℓ since there is a position of X that cannot be covered by a cyclic shift of the current $X[0..\ell - 1]$. If *covered* reaches $n - 1$ then include the current value of ℓ in the output and go to the next ℓ.

The correctness of the method can be seen as follows. For any fixed ℓ, if the output contains ℓ then every position of X belongs to at least one occurrence of a cyclic shift of $X[0..\ell - 1]$, and so $X[0..\ell - 1]$ is a cyclic cover of X by definition. On the other hand, if the output does not contain ℓ then there is some position i in X for which *covered* could never reach the value i; the only way that this can happen is if none of $X[i - \ell + 1..i]$, $X[i - \ell + 2..i + 1]$, ..., $X[i..i + \ell - 1]$ are cyclic shifts of $X[0..\ell - 1]$, which means that $X[0..\ell - 1]$ cannot be a cyclic cover of X. Furthermore, in this case, Lemma 1 guarantees that X does not have any cyclic cover of length ℓ.

According to Lemma 4, the preprocessing takes $\mathcal{O}(n)$ time and verifying each $\ell \in \{1, \ldots, n-1\}$ in the main for-loop takes $\mathcal{O}(n - \ell + 1)$ time. Thus, the total time complexity of the straightforward algorithm is $\mathcal{O}(n^2)$. The next section will present a faster solution.

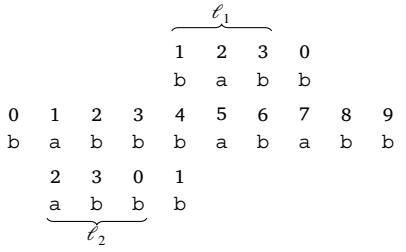

Fig. 4. Given X = babbbababb, W = bbab, and the constraint that $W[1]$ aligns to $X[4]$, the region $X[1..6]$ is cyclically covered by W. The lengths ℓ_1 and ℓ_2 denote $lcp(W[1..3]W[0], X[4..9])$ and $lcp^r(W[2..3]W[0..1], X[0..4]) - 1$, respectively.

3. A faster algorithm for cyclic covers

In order to improve the quadratic-time algorithm, we consider two cases: (i) if W is a highly periodic factor and (ii) if W is not a highly periodic factor. This is because the number of occurrences of a highly periodic factor W in a string X is much higher due to the repetitive nature permitting an overlap between each period of W; this is not the case if W is non-highly periodic.

Below, we outline an improved approach for the cyclic covers problem.

1. Section 3.1 presents a function named $FindFixedCover(W, X, i, j)$ which returns a region in X (if any) that is cyclically covered by W under the constraint that $W[i]$ aligns to $X[j]$.
2. Using the function $FindFixedCover(W, X, i, j)$, Section 3.2 develops an $\mathcal{O}(n/\ell)$-time algorithm for finding regions in X covered by W when W is highly-periodic.
3. Again using $FindFixedCover(W, X, i, j)$, Section 3.3 develops an $\mathcal{O}(n/\ell)$-time algorithm for finding regions in X covered by W when W is not highly-periodic.
4. Finally, in Section 3.4, we present how to identify all cyclic covers in $\mathcal{O}(n \log n)$ time.

3.1. The function $FindFixedCover(W, X, i, j)$

Let $X[0..n-1]$ be a string, $W[0..\ell - 1]$ a factor of X, and i and j nonnegative integers. 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 $X[j'..j' + \ell - 1]$. For example, given the string X = babbbababb, the factor W = bbab, and the constraint that $W[1]$ aligns to $X[4]$, the region $X[1..6]$ is cyclically covered by W. See Fig. 4.

The lemma below explains how to identify a region $X[\alpha..\beta]$ in X that is cyclically covered by W under the constraint that $W[i]$ aligns to $X[j]$, or determine that none exists.

Lemma 6. *Consider a string $X[0..n-1]$ and a length-ℓ factor $W[0..\ell - 1]$ of X. 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 $\ell_1 + \ell_2 \ge \ell$ then $X[j - \ell_2..j + \ell_1 - 1]$ is cyclically covered by W under the constraint that $W[i]$ aligns to $X[j]$; otherwise, such a cyclic cover does not exist.*

Proof. Let $U = W[i..\ell - 1]W[0..i-1]$ and define $U^2 = UU$. From the definitions of ℓ_1 and ℓ_2, we have $X[j - \ell_2..j + \ell_1 - 1] = U^2[\ell - \ell_2..\ell + \ell_1 - 1]$. Every factor of length ℓ (if any) of this string is a cyclic shift of U, and hence also of W. Thus, if $\ell_1 + \ell_2 \ge \ell$ then $X[j - \ell_2..j + \ell_1 - 1]$ is cyclically covered by W under the constraint that $U[0]$ aligns to $X[j]$ (i.e., $W[i]$ aligns to $X[j]$).

In contrast, if $\ell_1 + \ell_2 < \ell$ then the length of $X[j - \ell_2..j + \ell_1 - 1]$ is $(j + \ell_1 - 1) - (j - \ell_2) + 1 = \ell_1 + \ell_2 < \ell$. Then $|U| = \ell$ implies that if X contains a cyclic shift of U with $U[0]$ aligned to $X[j]$ then at least

one of $X[j-\ell_2-1]$ and $X[j+\ell_1]$ must belong to it. However, this is impossible because $U[\ell-\ell_2-1] \neq X[j-\ell_2-1]$ and $U[\ell_1] \neq X[j+\ell_1]$ by the definitions of ℓ_1 and ℓ_2. Therefore, X cannot contain a cyclic shift of W under the constraint that $W[i]$ aligns to $X[j]$. □

Based on Lemma 6, we define a function $FindFixedCover(W, X, i, j)$ which returns a pair of indices (α, β) with $\alpha \leq j \leq \beta$ such that $X[\alpha..\beta]$ is cyclically covered by W under the constraint that $W[i]$ aligns to $X[j]$. If no such cyclic cover exists, the function returns an empty region.

Lemma 7. *After $\mathcal{O}(n)$ time preprocessing, $FindFixedCover(W, X, i, j)$ for any factor W of X and any nonnegative integers i and j can be computed in $\mathcal{O}(1)$ time.*

Proof. In the preprocessing step, build the lcp and lcp^r data structures [23] for X in $\mathcal{O}(n)$ time. Then, for any call to $FindFixedCover(W, X, i, j)$, use them to compute ℓ_1 and ℓ_2 defined in Lemma 6 in $\mathcal{O}(1)$ time, and check if $\ell_1 + \ell_2 \geq \ell$. In accordance with Lemma 6, if the answer is yes then return $(j-\ell_2, j+\ell_1-1)$; if the answer is no then return $\emptyset$.

Since $W[i..\ell-1]W[0..i-1]$ may occur as a factor in X, we cannot always compute ℓ_1 with a single query to the lcp data structure for X. Instead, we compute ℓ_1 using at most two lcp queries as follows. If $lcp(W[i..\ell-1], X[j..n-1]) < \ell - i$ then it represents the sought value of ℓ_1 and we are done. Otherwise, $lcp(W[i..\ell-1], X[j..n-1]) = \ell - i$ indicates a potential longer match involving the cyclic shift $W[0..i-1]$. To determine the full extent of this match, we perform a second lcp query between $W[0..i-1]$ and $X[j+\ell-i..n-1]$. Thus the combined length of the match, $\ell_1 = (\ell - i) + lcp(W[0..i-1], X[j+\ell-i..n-1])$. Note that $\ell_1 \leq \ell$, where $\ell_1 = \ell$ corresponds to a complete occurrence of the cyclic shift $W[i..\ell-1]W[0..i-1]$ aligning with $X[j..j+\ell-1]$. The value of ℓ_2 is computed analogously. □

3.2. Finding regions in X that are cyclically covered by a highly-periodic factor W

Lemma 9 below describes how to find regions that are cyclically covered by $W[0..\ell-1]$ if W is of period q where $q \leq \ell/4$. To prove it, we make use of a lemma by Miyazaki et al. from [33] (see also [10,20]) to represent occurrences in a convenient way. Let (j_1, q, m) denote the arithmetic progression $j_1, j_2, \ldots, j_m$ with $j_{s+1} = j_s + q$, where $1 \leq s < m$.

Lemma 8 *([33], Lemma 3.1). Suppose the minimum period of $W[0..\ell-1]$ 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 9. *Suppose the smallest period of $W[0..\ell-1]$ 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 2). To see why $\ell/3 < |Y|$, observe that $\ell = 2qk + i$, where $k \in \mathbb{Z}^+, k \geq 2$ and $i \in [0, 2q-1]$. Here, we consider the edge case when $i = 2q-1$. It follows that

$$
\begin{aligned}
|Y| &= \left\lfloor \frac{2qk + 2q - 1}{2q} \right\rfloor q \\
&= \frac{2qk}{2q} q = kq \\
&> \frac{1}{3}(2qk + 2q - 1) \quad \text{(when } k \geq 2\text{)} \\
&= \frac{\ell}{3}
\end{aligned}
$$

as required.

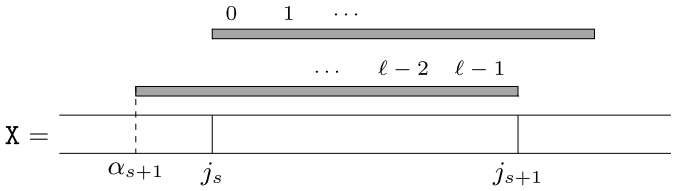

Fig. 5. If $\alpha_{s+1} < \alpha_s \leq j_s$ for $s \geq 1$, then $X[j_s..j_{s+1}]$ is a prefix of W occurring at j_s, and is a suffix of $LCP^r(W, X[0..j_{s+1}]) - 1$.

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 3, these occurrences can be found in $\mathcal{O}(1)$ time by computing $occ(Y, X[i'..i'+2|Y|])$ for $i' \in \{i - \ell + h|Y| \mid h = 0, 1, 2, \ldots, \lfloor 3\ell/|Y| \rfloor\}$. Since $3\ell/|Y| < 9$ we have at most 9 arithmetic progressions with period q (by Lemma 8) plus up to 18 standalone occurrences (i.e., occurrences which are not part of an arithmetic progression).

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 $X[\alpha_s..\beta_s] = FindFixedCover(W, X, 0, j_s)$. We claim that the following inequalities hold: $\beta_s \leq \beta_{s+1}$ and $\alpha_s \leq \alpha_{s+1}$. The former inequality $\beta_s = j_s + lcp(W, X[j_s..n-1]) - 1 \leq j_s + lcp(W[0..q-1]W, X[j_s..n-1]) - 1 = j_s + q + lcp(W, X[j_{s+1}..n-1]) - 1 = \beta_{s+1}$ is simple to see as W is a prefix of $W[0..q-1]W$. For the latter inequality $\alpha_s \leq \alpha_{s+1}$, notice that if $|W|$ is a multiple of q, then we can 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 (see Fig. 5), which in turn would imply that $|W| - q$ is a period of W. By Lemma 2, we have $gcd(q, |W| - q) < q$ is a period of W, which is a contradiction. This completes the proof of the claim.

From the claim, it follows that the region obtained for this sequence is $X[\alpha_1..\beta_m]$, and only two calls to $FindFixedCover$ are needed. □

In conclusion, as factors $X[k\ell..(k+1)\ell-1]$ for $k \in [0, \lfloor \frac{n}{\ell} \rfloor - 1]$ and $X[n-\ell..n-1]$ contain all positions of X, Lemma 9 has the following corollary.

Corollary 1. After an $\mathcal{O}(n)$-time preprocessing of the string $X[0..n-1]$, for any highly-periodic factor $W[0..\ell-1]$, 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 next lemma states that factors of a string X that are not highly-periodic do not occur frequently in X.

Lemma 10. *Let X be a string of length n and W a length-ℓ non-highly-periodic factor of X. Then W has $\mathcal{O}(n/\ell)$ occurrences in X.*

Proof. By the definition of highly-periodic, all periods of W have to be greater than $\ell/4$. Then any two occurrences of W in X are at distance greater than $\ell/4$, so W must have fewer than n/ℓ occurrences in X. □

Let W be a factor of X and let W' be a factor of W. If a cyclic shift of W contains W', we call it a *W'-containing cyclic shift of W*.

Consider $W[0..\ell-1] = W_l W_r$ where $|W_l| = \lfloor \ell/2 \rfloor$. Lemma 11 shows how to find all regions in X covered by cyclic shifts of W.

Lemma 11. *Consider $W[0..\ell-1] = W_l W_r$ 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. Every cyclic shift of W must contain either W_l or W_r. Hence, the lemma follows. □

Next, we describe an algorithm that finds all regions in X covered by W_l-containing cyclic shifts of $W_l W_r$. All regions in X covered by W_r-containing cyclic shifts of $W_r W_l$ can be found by an analogous algorithm.

To find all regions in X covered by W_l-containing cyclic shifts of $W_l W_r$, 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 $X[0..n-1]$ by Lemma 10. Thus, we can find all these occurrences in $\mathcal{O}(n/\ell)$ time given the IPM data structure from Lemma 4. Then, by calling the function $FindFixedCover$ from Section 3.1, the regions in X covered by these W_l-containing cyclic shifts of W can be found in $\mathcal{O}(n/\ell)$ time.

For a highly periodic W_l, let $q_l \le \ell/8$ be its shortest period and let d_l be the length of 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 $W_{r'}W_{l'}$. Furthermore, a W_l-containing cyclic shift of W (d-cyclic shift of W for $d = 0$ or $d \ge |W_l|$) 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 $|W_l| \le d \le d_l$).

It remains to show that for a highly-periodic W_l, when W and $W_{r'}W_{l'}$ are not highly-periodic then $W_{l'}W[d_l]$ and $W_{r'}W_l$ are not highly-periodic as well. The next two lemmas handle this last case.

Lemma 12. *$W_{l'}W[d_l]$ is non-periodic (hence also non-highly-periodic).*

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

We observe that q' cannot be a multiple of q_l as in this case $W[d_l] = W[d_l - q'] = W[d_l - q_l]$, 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 $W_{l'}$. □

Lemma 13. *$W_{r'}W_l$ is not highly-periodic.*

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

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

3.4. The cyclic covers problem

Now, we are ready to define a function $FindCyclicCover(W_l, W_r, X, \ell)$ that returns all regions in X that are covered by W_l-containing cyclic shifts of a length ℓ factor W. This function is described in Algorithm 1.

Lemma 14 summarizes the time complexity of $FindCyclicCover(W_l, W_r, X, \ell)$.

Lemma 14. *Given the lcp, IPM and 2-Period data structures for X, we can compute $FindCyclicCover(W_l, W_r, X, \ell)$ (and $FindCyclicCover(W_r, W_l, X, \ell)$) in $\mathcal{O}(n/\ell)$ time.*

Algorithm 1 $FindCyclicCover(W_l, W_r, X, \ell)$.

Output: Regions in X covered by W_l-containing cyclic shifts of W

Calculate shortest period q_l of W_l
$W_{l'} = W[0..d_l - 1]$ and $W_{r'} = W[d_l..\ell-1]$.
If $W = W_l W_r$ or $W_{r'}W_{l'}$ is of period $\le \ell/4$, apply Corollary 1 to find the regions of X covered by W and return the answer.
$Ans = \emptyset$
if W_l is not highly-periodic **then**
    Find $m_l = j_1, \ldots, j_m$ such that $X[j_s..j_s + |W_l| - 1] = W_l$
    For each j_s, $Ans = Ans \cup FindFixedCover(W, X, 0, j_s)$
else
    Find $m_l = j_1, \ldots, j_m$ such that $X[j_s..j_s + d_l] = W_{l'}W[d_l]$
    For each j_s, $Ans = Ans \cup FindFixedCover(W, X, 0, j_s)$
    Find $m_l = j_1, \ldots, j_m$ such that $X[j_s..j_s + |W_{r'}W_l| - 1] = W_{r'}W_l$
    For each j_s, $Ans = Ans \cup FindFixedCover(W, X, d_l, j_s)$
end if
Return Ans

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 query the 2-Period data structure from Lemma 5. Later, with the use of a single lcp query ($lcp(X, X[q_l..n-1])$ 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 $\mathcal{O}(n/\ell)$ calls to $FindFixedCover$, which results in a total time complexity of $\mathcal{O}(n/\ell)$.

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 $FindCyclicCover(W_r, W_l, X, \ell)$, 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 7, where this problem has already been solved.

Another string which does not need to appear in X is $W_{r'}W_l$ (symmetrically $(W_r W_l)[0..d_r]$ used by $FindCyclicCover(W_r, W_l, X, \ell)$). 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. □

We now present our main result in Theorem 1.

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

Proof. We will now describe how we extend the algorithms $FindCyclicCover(W_l, W_r, X, \ell)$ and $FindCyclicCover(W_r, W_l, X, \ell)$ to verify if every position in X is covered by the regions returned in $\mathcal{O}(n/\ell)$ time.

In the preprocessing step, we construct the Internal Data Structure answering lcp, IPM, and 2-Period queries in $\mathcal{O}(n)$ time (Lem-

mas 3 and 5). For any fixed ℓ, let $W_l = X[0..\lfloor \ell/2 \rfloor - 1]$ and $W_r = X[\lfloor \ell/2 \rfloor .. \ell - 1]$.

For a given ℓ, we will identify occurrences of W_l-containing cyclic shifts of $W_l W_r$ and W_r-containing cyclic shifts of $W_r W_l$ (Lemma 11). We have the following three cases: If W is highly periodic, if W_l is highly periodic and if W_l is not highly periodic. This only requires a comparison between parts of W_l and W_r which takes $\mathcal{O}(1)$ time in total. If W is highly periodic, then we use Corollary 1 to identify occurrences of W in $\mathcal{O}(n/\ell)$ time. Else, if W_l is not highly periodic, then we identify all occurrences of $X[j_s..j_s + |W_l| - 1] = W_l$ and store it in an array A_l using $\mathcal{O}(n/\ell)$ time (Lemma 10).

Otherwise if W_l is highly periodic, we compute its shortest period, q_l and the length of the longest prefix of W which is q_l-periodic (d_l). If $W_{r'} W_{l'} (= W[d_l..\ell - 1] W[0..d_l - 1])$ is highly periodic, then we calculate occurrences in $\mathcal{O}(n/\ell)$ time using Corollary 1.

Otherwise if $W_{r'} W_{l'}$ is not highly periodic, then we identify all occurrences of $X[j_s..j_s + d_l] = W_{l'} W[d_l]$ (Lemma 12) and store it in an array A_{l1}. We also identify all occurrences of $X[j_s..j_s + |W_{r'} W_l| - 1] = W_{r'} W_l$ (Lemmas 13 and 14) and store it in an array A_{l2}. As each substring is not highly periodic, there are at most n/ℓ occurrences in X and we can use Lemma 3 to identify all occurrences in $\mathcal{O}(n/\ell)$ time. The same method is repeated to identify W_r-containing cyclic shifts and also takes $\mathcal{O}(n/\ell)$ time.

For each of the arrays A_l (if W_l is not highly periodic) or A_{l1} and A_{l2} (if W_l is highly periodic), and similarly A_{r_1} and A_{r2} or A_r (depending on if W_r is highly periodic or not), we initialize variables $first_l$, $first_{l1}$, $first_{l2}$, $first_{r1}$, $first_{r2}$ and $first_r$, and set them to the first element of each array, in $\mathcal{O}(1)$ time. Each of these variables is used to keep track of the W_l or W_r-containing regions we have computed. Next, we compare $first_l$ (or $first_{l1}$ and $first_{l2}$) with $first_r$ (or $first_{r1}$ and $first_{r2}$), to identify the next occurrence of W_l or W_r in X, and perform an instance of $FindFixedCover()$. The value of $first_l/first_{l1}/first_{l2}/first_r/first_{r1}/first_{r2}$ is updated to next position in its array. We must also check that the leftmost position of the region returned by $FindFixedCover()$ is left to the rightmost position of the string that has been covered so far, which takes $\mathcal{O}(1)$ time to check. If at any point, we find a region that does not overlap with the covered region, then we return that X does not have a cyclic cover of length ℓ. This part of checking and traversing arrays is repeated until we have examined and performed $FindFixedCover()$ for all occurrences in each array. It will take $\mathcal{O}(n/\ell)$ time to traverse each array, as there are at most 4 arrays to traverse, where each array contains at most n/ℓ occurrences by Lemmas 12 and 13. It will also take $\mathcal{O}(n/\ell)$ time to compute $FindFixedCover()$ for all occurrences in each array.

The total time to test all $\ell = 1, \ldots, n$ is upper-bounded by $\mathcal{O}(\sum_{\ell=1}^{n} \frac{n}{\ell}) = \mathcal{O}(n \log n)$, using the asymptotic formula for the nth harmonic number. □

4. Concluding remarks

In this article, we introduced the problem of finding all the distinct cyclic covers of a string of length n and gave an algorithm for solving it in $\mathcal{O}(n \log n)$ time (Theorem 1). We conclude the article by defining two other closely related problems that can be solved even more efficiently.

4.1. Cyclic borders

A prefix of the form $X[0..\ell - 1]$ of a string X is a *cyclic border of* X if it is a cyclic shift of the suffix $X[n - \ell..n - 1]$. See Fig. 6 for an illustration.

The *cyclic borders problem* is to output the lengths of all prefixes of an input string X that are cyclic borders of X. The cyclic borders problem is solvable in $\mathcal{O}(n)$ time, where $n = |X|$, by using Lemma 4: First construct the IPM data structure for X in $\mathcal{O}(n)$ time, and then just check for each $\ell = 1, \ldots, n$ in $\mathcal{O}(1)$ time if the prefix $X[0..\ell - 1]$ is a cyclic shift of the corresponding suffix $X[n - \ell..n - 1]$.

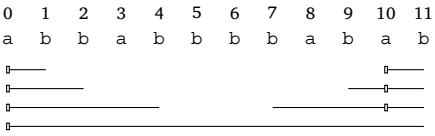

Fig. 6. The string X = abbabbbbabab has cyclic borders of lengths 2, 3, 5, and 12 because $X[10..11]$ = ab = $X[0..1]$, $X[9..11]$ = bab is a cyclic shift of $X[0..2]$ = abb, $X[7..11]$ = babab is a cyclic shift of $X[0..4]$ = abbab, and the whole string X is a cyclic border of itself.

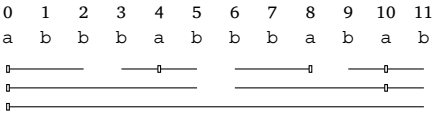

Fig. 7. The string X = abbbabbbabab has cyclic periods 3, 6, and 12. Here, X is 3-cyclic periodic because it can be divided into four consecutive substrings that are cyclic shifts of the factor abb. X is also 6-cyclic periodic since it can be broken into two consecutive substrings that are both circular factors of abbbab, and trivially 12-cyclic periodic.

Theorem 2 *(Cyclic borders problem). Given a string X of length n over an integer alphabet, we can find all integers $\ell > 0$ such that the prefix $X[0..\ell - 1]$ is a cyclic border of X in $\mathcal{O}(n)$ total time.*

4.2. Cyclic factorization

A *cyclic factorization* of a string X is a partition of X into factors of equal length such that each resulting factor is a cyclic shift of all the others. If there exists a cyclic factorization of X in which the factors have length ℓ then ℓ is called a *cyclic period of* X and X is called *ℓ-cyclic periodic*. For an example, see Fig. 7.

The *cyclic factorization problem* takes as input a string X and asks for all of the cyclic periods of X. It can be solved as follows. Construct the IPM data structure for X in $\mathcal{O}(n)$ time in a preprocessing step, and then, for every ℓ that divides n (written as $\ell \mid n$) if all of the $\frac{n}{\ell}$ consecutive nonoverlapping length-ℓ factors of X are cyclic shifts of $X[0..\ell - 1]$, include ℓ in the output. By Lemma 4, the preprocessing takes $\mathcal{O}(n)$ time, after which checking if any pair of factors are cyclic shifts of each other takes $\mathcal{O}(1)$ time. In total, the time complexity is $\mathcal{O}(n) + \mathcal{O}(\sum_{\ell \mid n} \frac{n}{\ell} \cdot 1) = \mathcal{O}(n \log \log n)$, due to the bound $\sum_{\ell \mid n} \frac{n}{\ell} = \mathcal{O}(n \log \log n)$ from [15, Equation (13)].

Theorem 3 *(Cyclic factorization problem). Given a string X of length n over an integer alphabet, we can find all integers $\ell > 0$ such that X is ℓ-cyclic periodic in $\mathcal{O}(n \log \log n)$ total time.*

CRediT authorship contribution statement

Roberto Grossi: Writing – review & editing, Writing – original draft, Supervision, Formal analysis, Conceptualization. **Costas S. Iliopoulos:** Writing – review & editing, Writing – original draft, Supervision, Formal analysis, Conceptualization. **Jesper Jansson:** Writing – review & editing, Writing – original draft, Supervision, Formal analysis, Conceptualization. **Zara Lim:** Writing – review & editing, Writing – original draft, Formal analysis. **Wing-Kin Sung:** Writing – review & editing, Writing – original draft, Supervision, Formal analysis, Conceptualization. **Wiktor Zuba:** Writing – review & editing, Writing – original draft, Supervision, Formal analysis.

Addendum

After the completion of the work in this article, a faster solution to the cyclic covers problem was discovered by Iliopoulos et al. [22]. They developed a new, non-trivial data structure for efficiently answering internal circular pattern matching queries which generalizes the IPM

data structure from [27] and that can be used to solve the cyclic covers problem in optimal $\mathcal{O}(n)$ time. Although their algorithm requires strengthened versions of our Lemmas 6, 7, and 14, its fundamental strategy is quite different from the one used here, as it relies on computing families of important short substrings and applying the algorithm recursively. In addition, Iliopoulos et al. [22] gave a faster algorithm for the cyclic factorization problem that runs in $\mathcal{O}(n)$ time. Their refined method still checks if ℓ is a cyclic period for each ℓ that divides n like the method in Theorem 3 above, but it tests candidate covers of short lengths more efficiently via deterministic substring hashing and counting.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

No data was used for the research described in the article.

References

[1] M. Alzamel, A. Conte, S. Denzumi, R. Grossi, C. Iliopoulos, K. Kurita, K. Wasa, Finding the anticover of a string, in: 31st Annual Symposium on Combinatorial Pattern Matching (CPM 2020), vol. 161, 2020.

[2] M. Alzamel, A. Conte, D. Greco, V. Guerrini, C. Iliopoulos, N. Pisanti, N. Prezza, G. Punzi, G. Rosone, Online algorithms on antipowers and antiperiods, in: International Symposium on String Processing and Information Retrieval, Springer, 2019, pp. 175–188.

[3] A. Amir, A. Levy, R. Lubin, E. Porat, Approximate cover of strings, Theor. Comput. Sci. 793 (2019) 59–69, https://doi.org/10.1016/j.tcs.2019.05.020.

[4] A. Apostolico, M. Crochemore, Fast parallel Lyndon factorization with applications, Math. Syst. Theory 28 (2) (1995) 89–108.

[5] A. Apostolico, A. Ehrenfeucht, Efficient detection of quasiperiodicities in strings, Theor. Comput. Sci. 119 (2) (1993) 247–265.

[6] A. Apostolico, M. Farach, C.S. Iliopoulos, Optimal superprimitivity testing for strings, Inf. Process. Lett. 39 (1) (1991) 17–20.

[7] L.A. Ayad, C. Barton, S.P. Pissis, A faster and more accurate heuristic for cyclic edit distance computation, Pattern Recognit. Lett. 88 (2017) 81–87.

[8] K.S. Booth, Lexicographically least circular substrings, Inf. Process. Lett. 10 (4–5) (1980) 240–242.

[9] D. Breslauer, An on-line string superprimitivity test, Inf. Process. Lett. 44 (6) (1992) 345–347.

[10] D. Breslauer, Z. Galil, Real-time streaming string-matching, ACM Trans. Algorithms 10 (4) (August 2014), https://doi.org/10.1145/2635814.

[11] A. Černỳ, Lyndon factorization of generalized words of Thue, Discret. Math. Theor. Comput. Sci. 5 (2002) 17–46.

[12] A. Conte, R. Grossi, A. Marino, Large-scale clique cover of real-world networks, Inf. Comput. 270 (2020) 104464.

[13] M. Crochemore, C.S. Iliopoulos, J. Radoszewski, W. Rytter, J. Straszyński, T. Waleń, W. Zuba, Shortest covers of all cyclic shifts of a string, Theor. Comput. Sci. 866 (2021) 70–81.

[14] M. Crochemore, W. Rytter, Jewels of Stringology: Text Algorithms, World Scientific, Singapore, 2002.

[15] R.L. Duncan, Some estimates for $\sigma(n)$, Am. Math. Mon. 74 (6) (1967) 713–715.

[16] K.A. Dunne, R.R. Chaudhuri, A.E. Rossiter, I. Beriotto, D.F. Browning, D. Squire, A.F. Cunningham, J.A. Cole, N. Loman, I.R. Henderson, Sequencing a piece of history: complete genome sequence of the original Escherichia coli strain, Microb. Genom. 3 (3) (2017) mgen000106.

[17] J.P. Duval, Factorizing words over an ordered alphabet, J. Algorithms 4 (4) (1983) 363–381.

[18] N.J. Fine, H.S. Wilf, Uniqueness theorems for periodic functions, Proc. Am. Math. Soc. 16 (1) (1965) 109–114, https://doi.org/10.2307/2034009.

[19] H. Fredricksen, J. Maiorana, Necklaces of beads in k colors and k-ary de Bruijn sequences, Discrete Math. 23 (3) (1978) 207–210.

[20] Z. Galil, Optimal parallel algorithms for string matching, Inf. Control 67 (1–3) (1985) 144–157, https://doi.org/10.1016/S0019-9958(85)80031-0.

[21] D. Gusfield, Algorithms on strings, trees, and sequences: computer science and computational biology, ACM SIGACT News 28 (4) (1997) 41–60.

[22] C. Iliopoulos, T. Kociumaka, J. Radoszewski, W. Rytter, T. Waleń, W. Zuba, Linear-time computation of cyclic roots and cyclic covers of a string, in: 34th Annual Symposium on Combinatorial Pattern Matching (CPM 2023), in: Leibniz International Proceedings in Informatics (LIPIcs), vol. 259, 2023, 15.

[23] J. Kärkkäinen, P. Sanders, Simple linear work suffix array construction, in: J.C.M. Baeten, J.K. Lenstra, J. Parrow, G.J. Woeginger (Eds.), Automata, Languages and Programming, 30th International Colloquium, ICALP 2003, Proceedings, Eindhoven, the Netherlands, June 30 – July 4, 2003, in: Lecture Notes in Computer Science, vol. 2719, Springer, 2003, pp. 943–955.

[24] T. Kociumaka, Efficient data structures for internal queries in texts, Ph.D. thesis, University of Warsaw, Poland, 2018, https://www.mimuw.edu.pl/~kociumaka/files/phd.pdf.

[25] T. Kociumaka, M. Kubica, J. Radoszewski, W. Rytter, T. Waleń, A linear time algorithm for seeds computation, in: Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2012), SIAM, 2012, pp. 1095–1112.

[26] T. Kociumaka, S.P. Pissis, J. Radoszewski, W. Rytter, T. Walen, Fast algorithm for partial covers in words, Algorithmica 73 (1) (2015) 217–233, https://doi.org/10.1007/s00453-014-9915-3.

[27] T. Kociumaka, J. Radoszewski, W. Rytter, T. Waleń, Internal pattern matching queries in a text and applications, in: Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2015), SIAM, 2015, pp. 532–551.

[28] Y. Li, W.F. Smyth, Computing the cover array in linear time, Algorithmica 32 (1) (2002) 95–106.

[29] M. Lothaire, Applied Combinatorics on Words, Encyclopedia of Mathematics and Its Applications, Cambridge University Press, 2005.

[30] M. Lothaire, Algebraic Combinatorics on Words, vol. 90, Cambridge University Press, New York, 2002.

[31] G. Melançon, Lyndon factorization of infinite words, in: Annual Symposium on Theoretical Aspects of Computer Science, Springer, 1996, pp. 147–154.

[32] G. Melançon, Lyndon factorization of Sturmian words, Discrete Math. 210 (1–3) (2000) 137–149.

[33] M. Miyazaki, A. Shinohara, M. Takeda, An improved pattern matching algorithm for strings in terms of straight-line programs, in: Annual Symposium on Combinatorial Pattern Matching, Springer, 1997, pp. 1–11.

[34] D. Moore, W.F. Smyth, An optimal algorithm to compute all the covers of a string, Inf. Process. Lett. 50 (5) (1994) 239–246.

[35] R.Z. Norman, M.O. Rabin, An algorithm for a minimum cover of a graph, Proc. Am. Math. Soc. 10 (2) (1959) 315–319.

[36] V. Palazón-González, A. Marzal, On the dynamic time warping of cyclic sequences for shape retrieval, Image Vis. Comput. 30 (12) (2012) 978–990.

[37] Y. Shiloach, Fast canonization of circular strings, J. Algorithms 2 (2) (1981) 107–121.

[38] M.J. Tisza, D.V. Pastrana, N.L. Welch, B. Stewart, A. Peretti, G.J. Starrett, Y.Y.S. Pang, S.R. Krishnamurthy, P.A. Pesavento, D.H. McDermott, et al., Discovery of several thousand highly diverse circular DNA viruses, eLife 9 (2020).

[39] E.K. Wagner, M.J. Hewlett, D.C. Bloom, D. Camerini, Basic Virology, vol. 3, Blackwell Science Malden, MA, 1999.

[40] D.J. Wurpel, S.A. Beatson, M. Totsika, N.K. Petty, M.A. Schembri, Chaperone-usher fimbriae of Escherichia coli, PLoS ONE 8 (1) (2013) e52835.