Theoretical Computer Science 866 (2021) 70–81

Contents lists available at ScienceDirect

Theoretical Computer Science

www.elsevier.com/locate/tcs

Shortest covers of all cyclic shifts of a string

Maxime Crochemore[a], Costas S. Iliopoulos[a], Jakub Radoszewski[b,∗,1], Wojciech Rytter[b], Juliusz Straszyński[b,1], Tomasz Waleń[b,1], Wiktor Zuba[b,1]

[a] *Department of Informatics, King's College London, London, UK*
[b] *Institute of Informatics, University of Warsaw, Warsaw, Poland*

A R T I C L E   I N F O

Article history:
Received 30 June 2020
Received in revised form 20 February 2021
Accepted 4 March 2021
Available online 17 March 2021

Keywords:
Cover
Quasiperiod
Seed
Fibonacci string

A B S T R A C T

A factor C of a string S is called a cover of S, if each position of S is contained in an occurrence of C. Breslauer (1992) [3] proposed a well-known $\mathcal{O}(n)$-time algorithm that computes the shortest cover of every prefix of a string of length n. We show an $\mathcal{O}(n \log n)$-time and $\mathcal{O}(n)$-space algorithm that computes the shortest cover of every cyclic shift of a string of length n and an $\mathcal{O}(n)$-time algorithm that computes the shortest among these covers. We also provide a combinatorial characterization of shortest covers of cyclic shifts of Fibonacci strings that leads to efficient algorithms for computing these covers.
We further consider the bound on the number of different lengths of shortest covers of cyclic shifts of the same string of length n. We show that this number is $\Theta(\log n)$ for Fibonacci strings.

1. Introduction

We consider strings as finite sequences of letters drawn from an alphabet $\Sigma = [0, n^{\mathcal{O}(1)}]$, often referred to as an integer alphabet [1]. The notion of periodicity in strings and its many variants have been well-studied in many fields like combinatorics on words, pattern matching, data compression, automata theory, formal language theory, and molecular biology. A typical regularity, the *period* U of a given string S, grasps the repetitiveness of S since S is a prefix of a string constructed by concatenations of U. If $S = AWB$, for some, possibly empty, strings A, W, B, then W is called a *factor* of S and, respectively, S is a superstring of W. A factor C of S is called a *cover* of S, if each position of S is contained in an occurrence of C. A factor C of S is called a *seed* of S, if there exists a superstring of S which is constructed by concatenations and superpositions of C. In other words, C is a seed of S if S is covered by occurrences and left and right overhangs of C. For example, *abc* is a period of *abcabcabca*, *abca* is a cover of *abcabcaabca*, and *abca* is a seed of *bcabcaabc*. The notions "cover" and "seed" are generalizations of periods in the sense that superpositions as well as concatenations are considered to define them, whereas only concatenations are considered for periods.

In computation of covers, two problems have been considered in the literature. The shortest-cover problem (also known as the superprimitivity test) is that of computing the shortest cover of a given string of length n, and the all-covers problem is that of computing all the covers of a given string. Apostolico et al. [2] introduced the notion of covers and gave a linear-time algorithm for the shortest-cover problem. Breslauer [3] proposed an on-line algorithm for computing the shortest cover that works in linear time. In particular, his algorithm computes the shortest cover of every prefix of a string. The

∗ Corresponding author.
E-mail addresses: maxime.crochemore@kcl.ac.uk (M. Crochemore), c.iliopoulos@kcl.ac.uk (C.S. Iliopoulos), jrad@mimuw.edu.pl (J. Radoszewski), rytter@mimuw.edu.pl (W. Rytter), jks@mimuw.edu.pl (J. Straszyński), walen@mimuw.edu.pl (T. Waleń), w.zuba@mimuw.edu.pl (W. Zuba).
[1] Supported by the Polish National Science Center, grant no. 2018/31/D/ST6/03991.

https://doi.org/10.1016/j.tcs.2021.03.011
0304-3975/© 2021 Elsevier B.V. All rights reserved.

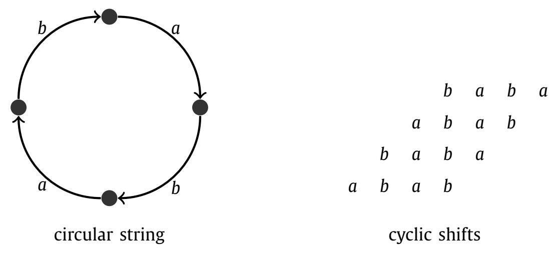


Fig. 1. The string *aba* is a cover of the string $S = abab$ treated as a single circular string, but is not a cover of any of cyclic shifts of S.

other direction was taken by Moore and Smyth [4,5] and by Li and Smyth [6] who computed all the covers of a string and a representation of all the covers of all prefixes of a string, respectively. A circular string S' corresponding to a given string S is formed by concatenating the first letter of S to the right of its last letter. Covers of circular strings were also considered. It is implicit in [7] that covers of a circular string S are exactly seeds of S^2. Covers and seeds of Fibonacci strings were studied in [8], whereas covers of circular Fibonacci strings were considered in [9].

All the seeds of a string of length n can be represented in $\mathcal{O}(n)$ space as a collection of a linear number of disjoint paths in the suffix trees of the string and of its reversal. This representation can be computed in $\mathcal{O}(n \log n)$ time [7] and even in $\mathcal{O}(n)$-time [10]. Recently it was also shown in [11] that all the seeds can also be represented as a linear number of disjoint paths in just the suffix tree of the string. This implies the following fact:

Lemma 1. *The problem of computing the shortest cover of a circular string can be solved in linear time.*

We say that a string Y is a *cyclic shift* of a string X if $X = AB$ and $Y = BA$ for some strings A and B; in this case we also write $Y = rot_{|A|}(X)$. It seems that the problem of computing shortest covers of all cyclic shifts of a string is harder than that of computing the shortest cover of a circular string. A straightforward application of any of the aforementioned algorithms for computing covers of a string yields an $\mathcal{O}(n^2)$-time solution to the problem. One should note that covers of circular strings are a different notion than that of covers of cyclic shifts of a string; see Fig. 1.

The shortest covers of cyclic shifts of a string can behave rather irregularly. For example, the length of the shortest cover of $S = abaababababababababa$ equals 3, whereas the shortest cover of $rot_1(S)$ has length 18.

We consider the following problem.

Shortest Covers of All Cyclic Shifts of a String
Input: A string S of length n.
Output: The lengths of the shortest covers of all cyclic shifts of S.

Let S be a string of length n and $ShCov(S)$ denote the shortest cover of S. We introduce an array $\mathbf{CC}_S$ of length n such that $\mathbf{CC}_S[i] = |ShCov(rot_i(S))|$. Our main result is computing this array. We also denote

$$\mathbf{CCSet}(S) = \{\mathbf{CC}_S[i] : i = 0, \ldots, n-1\}.$$

Example 1. For the Fibonacci strings $S_1 = abaab$, $S_2 = abaababaabaab$ we have:

$$\mathbf{CC}_{S_1} = [5, 5, 5, 3, 5],\ \mathbf{CC}_{S_2} = [5, 5, 13, 3, \ldots]$$
$$\mathbf{CCSet}(S_1) = \{3, 5\},\ \mathbf{CCSet}(S_2) = \{3, 5, 8, 13\}.$$

Our results. We show that the whole array $\mathbf{CC}_S$ and $\min_i \mathbf{CC}_S[i]$ for a string S of length n can be computed in $\mathcal{O}(n \log n)$ time and $\mathcal{O}(n)$ time, respectively, and $\mathcal{O}(n)$ space. For this we use a characterization of covers of cyclic shifts of a string by seeds and squares, i.e., strings of the form W^2, and the suffix tree data structure.

We give a simple recursive formula for computing $\mathbf{CC}_S$ for a Fibonacci string S. It implies a linear-time algorithm for computing this array as a whole and can be used to devise time and space efficient algorithms for computing subsequent elements of this array. We also show that for the family of Fibonacci strings we have $|\mathbf{CCSet}(S)| = \Theta(\log |S|)$.

Structure of the paper. In Section 2 we recall basic definitions and illustrate them by showing a linear-time algorithm that solves a similar problem to the one in scope, that is, computing the shortest periods of all cyclic shifts of a string. Then in Section 3 we present characterizations of shortest covers of cyclic shifts of a string, which lead us to the main algorithmic results in Section 4. Shortest covers of cyclic shifts of Fibonacci strings are studied in Section 5. We conclude and mention some open problems in Section 6.

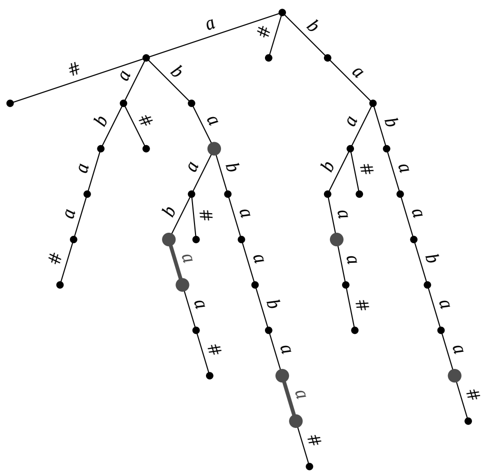


Fig. 2. The string $S = ababaabaa$ has the following seeds: *aba*, *abaab*, *baaba*, *abaaba*, *ababaaba*, *babaabaa*, *ababaabaa*. They can be represented on the (uncompressed) suffix tree of S as shown in the figure. Each seed is a path from root to marked node. In some cases, e.g. *abaab*, *abaaba*, multiple seeds are represented on a single path.

This is a full version of the paper [12]. In particular, compared to the conference version, it contains a much more precise characterization of shortest covers of cyclic shifts of Fibonacci strings.

2. Preliminaries

We assume that positions of a string S are numbered 0 through $|S|-1$, $S = S[0]\dots S[|S|-1]$. By $S[i\,..\,j]$ we denote a factor of S equal to $S[i]\dots S[j]$. A factor is called a prefix of $i = 0$ and a suffix if $j = |S|-1$. A factor that occurs both as a prefix and as a suffix of S is called a border of S. A positive integer p is a period of S if $S[i] = S[i+p]$ for all $i = 0,\dots,|S|-p-1$.

2.1. Suffix trees

Recall that a *suffix tree* of a string S is a compact tree of all the suffixes of $S\#$, where # is a special end marker. The root, branching nodes (i.e., nodes with more than one child), and leaves of the tree are *explicit*. All the remaining nodes are *implicit* in the tree. Each leaf is labeled with the starting position of the corresponding suffix. Every factor of S is represented as an explicit or implicit node of the tree. A suffix tree of a string of length n over an integer alphabet can be constructed in $\mathcal{O}(n)$ time [1].

Observation 1. Let S be a string of length n. After $\mathcal{O}(n)$-time preprocessing, all the occurrences of a factor of S, represented as a node in the suffix tree of S, can be reported in linear time w.r.t. the number of these occurrences.

Proof. It suffices to store a list L of leaves of the suffix tree in a left-to-right order. Then for every explicit node v of the tree, we precompute the endpoints of the sublist of L that corresponds to the occurrences of the string v. This precomputation is done bottom-up in $\mathcal{O}(n)$ time. □

We also use the following lemma.

Lemma 2 *([11]). Given a collection of factors $U_1,\dots,U_k$ of a string S of length n, each represented by an occurrence in S, in $\mathcal{O}(n+k)$ time we can compute the implicit or explicit node in the suffix tree of S that corresponds to each factor U_i. Moreover, all these nodes can be made explicit in $\mathcal{O}(n+k)$ time.*

The set (possibly of a quadratic size) of all seeds of a string can be represented as a collection of linearly many disjoint paths in the suffix tree [11]. It can be assumed that each path belongs to a single edge of the suffix tree. The endpoints of the paths can be implicit nodes. For an example, see Fig. 2.

2.2. Runs and squares

A string X is called *primitive* if $X = Y^k$ for positive integer k implies that $k = 1$. A string of the form Z^2 is called a *square*; it is called *primitively rooted* if Z is primitive.

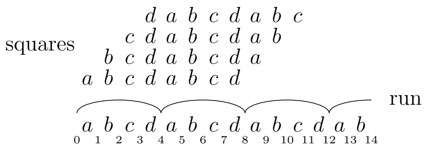


Fig. 3. Primitive squares can be derived from runs, knowing the shortest periods of runs. The leftmost square $(abcd)^2$ has its center at interposition 4.

i	$rot_i(S)$	min. period
0	*abababa*	2
1	*bababaa*	7
2	*ababaab*	5
3	*babaaba*	5
4	*abaabab*	5
5	*baababa*	5
6	*aababab*	7

Fig. 4. Smallest periods of all cyclic shifts of string $S = abababa$.

A *run* (also called a maximal repetition) in a string S is a triple (a, b, p) such that $S[a\,..\,b]$ has period p, $2p \le b - a + 1$ and the interval $[a, b]$ cannot be extended to the left nor to the right without violating the above property, that is, $S[a-1] \neq S[a+p-1]$ and $S[b-p+1] \neq S[b+1]$, provided that the respective positions exist. The *exponent* of a run (a, b, p) is defined as $\frac{b-a+1}{p}$. A string of length n has $\mathcal{O}(n)$ runs and they can all be computed in $\mathcal{O}(n)$ time [13,14].

From a run (a, b, p) we can produce all triples (a, b, kp) for integer $k \ge 1$ such that $2kp \le b - a + 1$; we call such triples *generalized runs*. That is, the period of a generalized run need not be the shortest period. The number of generalized runs is also $\mathcal{O}(n)$ as the sum of exponents of runs is $\mathcal{O}(n)$ [13,14].

We say that a square factor $S[i\,..\,i+2\ell-1]$ in S is *induced* by a generalized run (a, b, p) if $\ell = p$ and $[i, i+2\ell-1] \subseteq [a, b]$. A square factor is induced by exactly one generalized run and a primitively rooted square factor is induced by exactly one run [15]; see also Fig. 3.

An *interposition* i in S, for $i = 1, \ldots, |S| - 1$, is a delimiter between positions i and $i - 1$. Moreover, interposition 0 precedes the first letter of S and interposition $|S|$ follows the last letter. Thus a string S has $|S| + 1$ interpositions. We use interpositions to describe centers of square factors; see Fig. 3.

As an illustration of these notions, we show below that smallest periods of all cyclic shifts of a string can be computed in $\mathcal{O}(n)$ time. See also Fig. 4.

SMALLEST PERIODS OF ALL CYCLIC SHIFTS OF A STRING

Input: A string S of length n.

Output: The lengths of the smallest periods of all cyclic shifts of S.

Proposition 1. *Smallest periods of all cyclic shifts of a string of length n can be computed in $\mathcal{O}(n)$ time.*

Proof. Let S be a string of length n. We will show how to compute the smallest periods of strings of the form $rot_i(S)$ from the longest squares in the string $W = S^3$. Let $f_{W,k}(i)$ be the half length of the longest square with the center at the interposition i in W and half length smaller than k:

$$f_{W,k}(i) = \max\{j \in [0, k-1] : W[i-j\,..\,i-1] = W[i\,..\,i+j-1]\}.$$

We can observe that $f_{W,n}(n+i)$ is the longest border of $rot_i(S)$ (see Fig. 5), so the smallest period of $rot_i(S)$ is $n - f_{W,n}(n+i)$.

In [15] it was shown how to compute the shortest square centered at each interposition of a string in linear time from the runs in the string. The solution used a min-variant of a so-called Manhattan Skyline Problem. The array $f_{W,n}$ can be computed using a max-variant of the problem, stated formally below.

MAX-VARIANT OF MANHATTAN SKYLINE PROBLEM

Input: A set $\mathcal{I}$ of $\mathcal{O}(n)$ subintervals of $[0, 3n]$ with natural heights of size $\mathcal{O}(n)$

Output: The table $f[t] = \max\{height([i, j]) : t \in [i, j],\ [i, j] \in \mathcal{I}\},\ t \in [0, 3n]$.

The solution to the problem is obtained in the same way as it was shown in [15] for the min-variant.

Claim 1 *(See [15, Lemma 16]). The Max-Variant of the Manhattan Skyline Problem can be solved in linear time.*

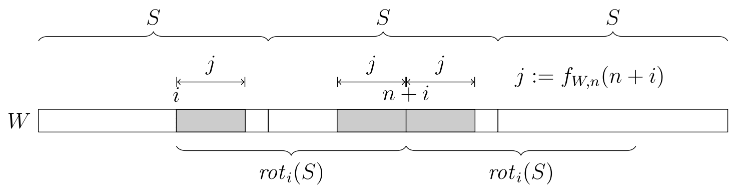


Fig. 5. Relation between $f_{W,n}(n+i)$ and the smallest period of $rot_i(S)$.

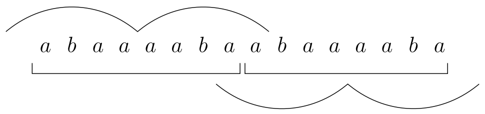


Fig. 6. Illustration of Example 2.

In our case subintervals correspond to generalized runs in S, since each generalized run (a,b,p) induces squares of half length p with centers at interpositions $\{a+p,\dots,b-p+1\}$. $\square$

Remark 1. In the above proof, taking $W=S^3$ (and not, say, $W=S^2$) is necessary when $f_{W,n}(n+i)$ is almost n.

3. Covers of cyclic shifts

We denote by $\frac{1}{2}$-*Squares*(S) (square halves) the set of factors Z of S such that the square Z^2 is also a factor of S and by $\frac{1}{2}$-*PSquares*(S) the subset of $\frac{1}{2}$-*Squares*(S) that consists only of primitive strings. We further denote by *Seeds*(S) the set of factors which are seeds of S. We use these sets for the string S^3 in order to characterize covers of all cyclic shifts of S.

Lemma 3. *Let S be a string of length n and C be a string of length up to n. Then C is a cover of $rot_i(S)$ if and only if $C \in Seeds(S^3) \cap \frac{1}{2}$-Squares$(S^3)$ and C^2 occurs with its center at interposition $j \equiv i \pmod n$ in S^3.*

Moreover, if C is the shortest cover of $rot_i(S)$, then $C \in Seeds(S^3) \cap \frac{1}{2}$-PSquares$(S^3)$.

Proof.

($\Rightarrow$) String C is a cover of $(rot_i(S))^4$, and thus a seed of its factor S^3. Moreover, $S^3[j-|C|\,..\,j+|C|-1]$, that is, the factor of S^3 of length $2|C|$ with center at interposition j, is equal to C^2 for $j=i+n$.

($\Leftarrow$) The square C^2 occurs in S^3 with its center at interposition $j \equiv i \pmod n$. Thus C is a border of $rot_j(S)=rot_i(S)$ as $|C|<n$. C is also a seed of $rot_i(S)$ which is a factor of S^3, hence it is a cover of $rot_i(S)$.

As for the "moreover" part, it suffices to note that the shortest cover of a string is obviously primitive. $\square$

Example 2. In the above lemma, one could not take S^2 instead of S^3. Indeed, for $S = abaaaaba$ we have that $rot_4(S) = aabaabaa$ has the shortest cover $aabaa$, but $S^2 = abaaaabaabaaaaba$ does not contain the square $(aabaa)^2$; see Fig. 6.

Let $T(S^3)$ be the suffix tree of S^3 in which we distinguish the nodes v corresponding to strings Z^2 for $Z \in Seeds(S^3) \cap \frac{1}{2}$-*PSquares*$(S^3)$. These nodes are called *candidate nodes*. Some of these nodes could be implicit nodes in the suffix tree. Then they are made explicit. Denote by *CandAnc*(v) the set of ancestor nodes of v in $T(S^3)$ which are candidate nodes. Let $|v|$ be the length of the string corresponding to the node v.

We can reformulate Lemma 3 as follows; see also Fig. 7.

Lemma 4. $\mathbf{CC}_S[i]$, *i.e., the length of the shortest cover of $rot_i(S)$, equals*

$$\min_{j,v}\{k\,:\,k=|v|/2,\ i=(j+k)\bmod n,\ j\in Leaves(T(S^3)),\ v\in CandAnc(j)\}.$$

Clearly if C is a cover of $rot_i(S)$, then C is a cover of S treated as a circular string. As we have already noted in Fig. 1, the converse is not necessarily true. However, we show that every shortest cover of the circular string S is a cover of the corresponding cyclic shift of S.

Lemma 5. *A shortest cover of a circular string is always a (shortest) cover of some cyclic shift.*

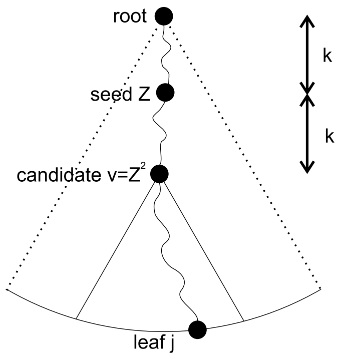


Fig. 7. Illustration of Lemma 4. The situation when $\mathbf{CC}_S[i] = k$. We have that $i = (j + k) \bmod n$ and Z^2 is a primitively rooted square of length $2k$; it corresponds to the node v which is possibly inside an edge of the suffix tree.

Proof. We need the following claim.

Claim 2 *(See [7]). String C is a cover of S considered as a circular string iff it is a seed of S^2, hence also iff it is a seed of S^3.*

Let C be a cover of a circular string S, such that C^2 does not occur in it. Consider the last position covered by any occurrence of C in the string.

The position must be also covered by another occurrence of C (the next position must be covered and cannot be the first position of some C). Thus by erasing the last position of C we obtain a shorter cover. Hence if C is a shortest cover then C^2 must appear in the circular string S.

By Lemma 3, C is a cover of some cyclic shift of S. □

By computing the shortest cover of the circular string S using Lemma 1 we obtain the following preliminary result.

Corollary 1. *For a string S of length n, $\min \mathbf{CC}_S$ can be computed in $\mathcal{O}(n)$ time.*

4. Main algorithm

First we have to show how to compute efficiently the tree $T(S^3)$. We denote by $\mathit{OccPSquares}(S)$ the set of all occurrences of primitively rooted squares in S. Each occurrence is represented in $\mathcal{O}(1)$ space as a factor of S. A direct consequence of the Three-square-prefix Lemma, see [16], is that a string of length n has no more than $\log n$ prefixes that are primitively rooted squares.

Lemma 6 *([16]). For a string S of length n, $|\mathit{OccPSquares}(S)| = \mathcal{O}(n \log n)$.*

Lemma 7. *For a string S of length n, $|\frac{1}{2}\text{-}\mathit{PSquares}(S)| = \mathcal{O}(n)$ and this set can be computed in $\mathcal{O}(n)$ time.*

Proof. Let us start with efficient computation of squares.

Claim 3 *([17,15,18–20]). For a string S of length n, we have $|\frac{1}{2}\text{-}\mathit{Squares}(S)| = \mathcal{O}(n)$ and this set can be computed in $\mathcal{O}(n)$ time.*

By the claim, $|\frac{1}{2}\text{-}\mathit{PSquares}(S)| = \mathcal{O}(n)$ since $\frac{1}{2}\text{-}\mathit{PSquares}(S) \subseteq \frac{1}{2}\text{-}\mathit{Squares}(S)$.

The set $\frac{1}{2}\text{-}\mathit{PSquares}(S)$ can be computed by filtering out the factors from $\frac{1}{2}\text{-}\mathit{Squares}(S)$ that are not primitive. This can be done in $\mathcal{O}(1)$ time per factor after $\mathcal{O}(n)$-space and time preprocessing using so-called Two-Period queries [14,21]. A more direct approach would be to (effortlessly) adapt the algorithm for computing different square factors from [15] using relations between primitive squares and *runs* (maximal repetitions); see Fig. 3. □

Lemma 8. *The tree $T(S^3)$ can be computed in $\mathcal{O}(n)$ time.*

Proof. We use a version of a minimal augmented suffix tree (MAST, in short), a data structure that was initially introduced in [22].

Let us recall that Lemma 2 can be used to augment the suffix tree of S^3 with nodes that correspond to a set of factors of S^3. We first apply the lemma to the collection of factors $\frac{1}{2}\text{-}\mathit{PSquares}(S^3)$, which can be efficiently computed due to the previous lemma.

Then we compute a representation of all the seeds of S^3 in the suffix tree using the algorithm from [11]. The representation consists of a collection of disjoint paths, each located on a single edge in the suffix tree; see Fig. 2. The endpoints of the paths that are implicit nodes can also be made explicit using Lemma 2.

For every node v corresponding to an element in $\frac{1}{2}$-*PSquares*(S^3), we check if it is located on some path that belongs to the representation of *Seeds*(S^3). Finally, we again use Lemma 2 for the original suffix tree of S^3 and set of factors Z^2 that correspond to all such elements $Z \in \frac{1}{2}\text{-}\mathit{PSquares}(S^3) \cap \mathit{Seeds}(S^3)$ to augment the suffix tree with the set of candidate nodes. This completes the proof. □

The number of integers equal to $(j+k) \bmod n$ is constant, hence we can forget about computing modulo n and for each $0 \le i < 3n$ we are to compute:

$$\min_{j,v}\{k \,:\, k=|v|/2,\ i=j+k,\ j\in \mathit{Leaves}(T(S^3)),\ v\in \mathit{CandAnc}(j)\}. \tag{1}$$

Algorithm 1: ComputeCC.

```
Initialize each entry of **CC** to $+\infty$
Compute $T(S^3)$
**foreach** *candidate node v in* $T(S^3)$ **do**
    **foreach** *occurrence* $S^3[j\,..\,j+|v|-1]$ *of v in* $S^3$ **do**
        $i := (j+|v|/2) \bmod n$
        $\mathbf{CC}[i] := \min(\mathbf{CC}[i],\ |v|/2)$
**return CC**
```

Theorem 1. *The algorithm ComputeCC computes the lengths of shortest covers for all cyclic shifts of a string in* $\mathcal{O}(n\log n)$ *time and* $\mathcal{O}(n)$ *space.*

Proof. In the third paragraph we simply implement (1). This proves correctness. It is possible to iterate over the occurrences of v in $\mathcal{O}(n)$ space using Observation 1. The required complexity follows from Lemmas 6 and 8. □

5. Shortest covers of cyclic shifts of Fibonacci strings

Recall that the Fibonacci strings are defined as $Fib_0 = b$, $Fib_1 = a$, $Fib_k = Fib_{k-1}Fib_{k-2}$ for $k \ge 2$. In other words, $Fib_k = \phi^k(Fib_0)$, where ϕ is a morphism

$$\phi(a) = ab, \quad \phi(b) = a.$$

Hence

$$Fib_2 = ab,\ Fib_3 = aba,\ Fib_4 = abaab,\ Fib_5 = abaababa,\ \ldots .$$

We denote $F_k = |Fib_k|$, the k-th Fibonacci number. In this section we give a precise characterization of **CC** and **CCSet** of Fibonacci strings. We further denote $\mathbf{CF}_n = \mathbf{CC}_{Fib_n}$. See Fig. 8 for an example.

Let $lcp(U, V)$ denote the length of the longest common prefix of strings U and V. We use the following known properties of Fibonacci strings.

Observation 2. $lcp(Fib_n, Fib_{n-2}Fib_{n-1}) = F_n - 2$.

Fact 1 *([23,24]). For every non-empty factor* U^2 *of* Fib_k, U *is a cyclic shift of* Fib_m *for some* m.

The following lemma bounds $\mathbf{CCSet}(Fib_n)$. Later we will show that actually $\mathbf{CCSet}(Fib_n) = \{F_3, \ldots, F_n\}$.

Lemma 9. $\mathbf{CCSet}(Fib_n) \subseteq \{F_1, \ldots, F_n\}$.

Proof. By Lemma 3, every element of the set $\mathbf{CCSet}(Fib_n)$ is a square half of length at most F_n in Fib_n^3. By Fact 1, the lengths of square halves in a Fibonacci string are Fibonacci numbers. It suffices to note that, for $n \ge 3$, Fib_n^3 is a factor of Fib_{n+5} since

$$Fib_8 = abaababaabaababaab\underline{abaabaaba}baabaab$$

contains a cube Fib_3^3 (underlined). □

Let us recall that there is a connection between covers of $rot_i(Fib_n)$ and seeds of Fib_n^3 (Lemma 3). The following lemma gives a combinatorial characterization of the latter.

$\mathbf{Fib}_2 = ab$

i	**cyclic shift**	**shortest cover**	**CF**$[i]$
0	*ab*	*ab*	2
1	*ba*	*ba*	2

$\mathbf{Fib}_3 = aba$

0	*aba*	*aba*	3
1	*baa*	*baa*	3
2	*aab*	*aab*	3

$\mathbf{Fib}_4 = abaab$

0	*abaab*	*abaab*	5
1	*baaba*	*baaba*	5
2	*aabab*	*aabab*	5
3	*ababa*	*aba*	3
4	*babaa*	*babaa*	5

$\mathbf{Fib}_5 = abaababa$

i	**cyclic shift**	**shortest cover**	$\mathbf{CF}_5[i]$
0	*abaababa*	*aba*	3
1	*baababaa*	*baababaa*	8
2	*aababaab*	*aababaab*	8
3	*ababaaba*	*aba*	3
4	*babaabaa*	*babaabaa*	8
5	*abaabaab*	*abaab*	5
6	*baabaaba*	*baaba*	5
7	*aabaabab*	*aabaabab*	8

$\mathbf{Fib}_6 = abaababaabaab$

i	**Fibonacci representation of** $i+1$	**cyclic shift**	**shortest cover**	$\mathbf{CF}_6[i]$
0	000001	*abaababaabaab*	*abaab*	5
1	000010	*baababaabaaba*	*baaba*	5
2	000100	*aababaabaabab*	*aababaabaabab*	13
3	000101	*ababaabaababa*	*aba*	3
4	001000	*babaabaababaa*	*babaabaababaa*	13
5	001001	*abaabaababaab*	*abaab*	5
6	001010	*baabaababaaba*	*baaba*	5
7	010000	*aabaababaabab*	*aabaababaabab*	13
8	010001	*abaababaababa*	*aba*	3
9	010010	*baababaababaa*	*baababaa*	8
10	010100	*aababaababaab*	*aababaab*	8
11	010101	*ababaababaaba*	*aba*	3
12	100000	*babaababaabaa*	*babaababaabaa*	13

Fig. 8. Shortest covers of cyclic shifts of Fibonacci strings.

Lemma 10. *Let $n \geq 2$.*

(a) If S is a seed of Fib_n^3 and $|S| \leq F_{n-1}$, then S is a seed of Fib_{n+1}^3.
(b) $rot_k(Fib_n)$ is a seed of Fib_{n+1}^3 for $0 \leq k \leq F_{n-1} - 2$.

Proof. (a) We will show that S is a cover of a factor U of Fib_{n+1}^2 of length at least F_{n+1}. This is sufficient to prove that S is a seed of Fib_{n+1}^3.

By Observation 2, we have that $lcp(Fib_n^3, Fib_{n+1}^2) = F_{n+2} - 2$; see Fig. 9. Let X be the set of occurrences of S in Fib_n^3 that start at the first $2F_n$ positions. Those occurrences cover the string $U = Fib_n^3[\min(X) \,..\, \max(X) + |S| - 1]$. Let us note that the first occurrence of S in the third Fib_n is at position $2F_n + \min(X)$ (possibly it is an overhang), so the position $2F_n + \min(X) - 1$ must be covered by the occurrence at position $\max(X)$. Therefore

$$\max(X) \geq 2F_n + \min(X) - |S|, \quad \text{hence} \quad |U| = \max(X) + |S| - \min(X) \geq 2F_n > F_{n+1}. \tag{2}$$

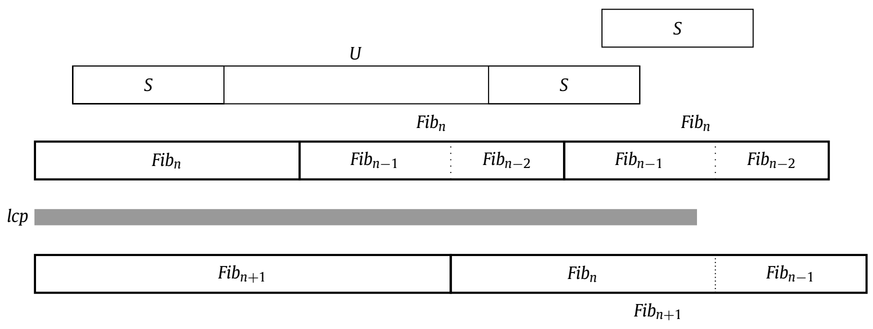


Fig. 9. $lcp(Fib_n^3, Fib_{n+1}^2) = F_{n+2} - 2$. Factor U and the first occurrence of S in Fib_n^3 outside its prefix Fib_n^2 are shown.

If $\max(X) + |S| - 1 < F_{n+2} - 2$, then U is a factor of Fib_{n+1}^2, which concludes the proof. Otherwise $|S| = F_{n-1}$ and $\max(X) = 2F_n - 1$; in this case it can be readily verified that S is not a seed of Fib_n^3.

(b) The string $S = rot_k(Fib_n)$ occurs at positions k and $F_n + k$ in Fib_n^3, covering a factor of length $2F_n > F_{n+1}$. It ends at position $2F_n + k - 1 < F_{n+2} - 2$, so it occurs also in Fib_{n+1}^2. Hence, S is a seed of Fib_{n+1}^3. □

Example 3. The upper bound on k in Lemma 10(b) is tight. E.g., $Fib_3 = aba$ is a seed of Fib_3^3 and of $Fib_4^3 = abaababaabababaab$, whereas $rot_1(Fib_3) = baa$ is a seed of Fib_3^3 but not of Fib_4^3.

For a sequence of numbers X let us denote by X^+ the sequence X with elements F_n changed to F_{n+1}.

Theorem 2. *For $n \geq 4$ we have:*

(a) $\mathbf{CF}_{n+1} = C^+B^+C^+AB^+$, *where* $\mathbf{CF}_n = ABC$, $|B| = F_{n-3}$, $|A| = |C| = F_{n-2}$.
(b) Let S_n denote the prefix of $\mathbf{CF}_n$ of length $F_{n-1} - 1$. Then

$$\mathbf{CF}_n = S_{n-2}\, F_n\, S_{n-3}\, F_n\, S_{n-2}\, F_n\, S_{n-1} F_n.$$

Proof. (a) The proof goes by induction. Two additional conditions are included in the statement:

- A and B do not contain F_{n-1} (this immediately follows from the fact that they are constructed from blocks with a $^+$);
- F_n occurs in $\mathbf{CF}_n$ only four times, at positions $F_{n-3} - 1, F_{n-2} - 1, F_{n-1} - 1, F_n - 1$ (that is, at ends of blocks A, B, C and once in the middle of the block A).

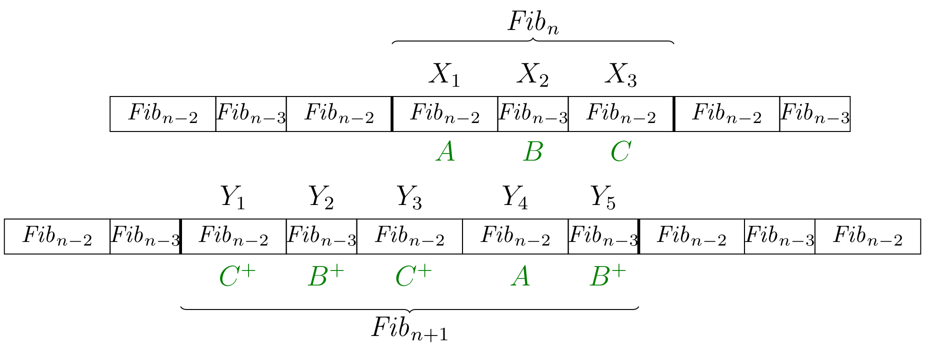
We use the following crucial observation that immediately follows from the characterization of covers of cyclic shifts (Lemma 3) and Lemma 10(a).

Observation 3. If $x = \mathbf{CF}_n[i]$ with $x \neq F_n$ and a string of length $2x$ with its center at interposition i in Fib_n (seen as a circular string) is the same as a string of length $2x$ with its center at interposition j in Fib_{n+1} (also circular), then $\mathbf{CF}_{n+1}[j] = x$.

Let us factorize strings Fib_n and Fib_{n+1} into named blocks, as shown in Fig. 10. We see that:

- Y_1 has the same surrounding of length $F_{n-1} - 2$ as X_3 ($lcp(Fib_{n-1}, Fib_{n-3}Fib_{n-2}) = F_{n-1} - 2$ by Observation 2).
- Y_2 has the same surrounding of length F_{n-2} as X_2 (Fib_{n-2} ends with a different letter than Fib_{n-3}).
- Y_3 has the same surrounding of length F_{n-1} as X_3 (Fib_{n-2} ends with a different letter than Fib_{n-3}).
- Y_4 has the same surrounding of length at least F_n as X_1 ($Fib_{n-3}Fib_{n-2}$ starts with Fib_{n-2}).
- Y_5 has the same surrounding of length $F_n - 2$ as X_2 (again by Observation 2).

For each of the blocks Y_3, Y_4, Y_5, the surrounding of length at least F_{n-1} is the same as for the corresponding X-block. By the observation, for each of these blocks, the parts of $\mathbf{CF}_{n+1}$ contain the same values $\leq F_{n-1}$ in the same places as their



Fib_n
X_1 X_2 X_3
Fib_{n-2} Fib_{n-3} Fib_{n-2} Fib_{n-2} Fib_{n-3} Fib_{n-2} Fib_{n-2} Fib_{n-3}
A B C
Y_1 Y_2 Y_3 Y_4 Y_5
Fib_{n-2} Fib_{n-3} Fib_{n-2} Fib_{n-3} Fib_{n-2} Fib_{n-2} Fib_{n-3} Fib_{n-2} Fib_{n-3} Fib_{n-2}
C^+ B^+ C^+ A B^+
Fib_{n+1}

Fig. 10. Factorizations of Fib_n and Fib_{n+1}.

corresponding X-blocks. The same also holds for blocks Y_1 and Y_2; in case of Y_1 we use the fact that the last position of C contains F_n, and in case of Y_2 that B does not contain F_{n-1}.

Now it remains to take care of the last six positions of $\mathbf{CF}_{n+1}$, the ones obtained from the four positions of $\mathbf{CF}_n$ that are equal to F_n. By Lemma 9 each of those positions has their $\mathbf{CF}_{n+1}$ value equal to either F_n or F_{n+1} (Lemma 10(a) works in both ways). For positions $F_{n-2}-1, F_{n-1}-1, F_n-1, F_{n+1}-1$ one can check that the strings of length $2F_n$ with their centers at the corresponding interpositions are not squares. For example, for position $F_{n-2}-1$ the string in question equals

$$aFib_{n-2}Fib_{n-3}Fib'_{n-2}\ aFib_{n-3}Fib_{n-2}Fib'_{n-2},$$

where a is the last letter of Fib_{n-2} and Fib'_n is Fib_n with the last letter removed; we have $Fib_{n-2}Fib_{n-3} \neq Fib_{n-3}Fib_{n-2}$ by Observation 2. Hence for those positions $\mathbf{CF}_{n+1}$ contains values F_{n+1}.

However, for positions $F_n + F_{n-3} - 1$ and $F_n + F_{n-2} - 1$ such strings match and are equal to $Fib_{n-4}Fib_{n-3}Fib_{n-1}$ shifted to the right by one and to $Fib_{n-3}Fib_{n-2}Fib_{n-1}$ shifted to the right by one, respectively. For example,

$$rot_{F_{n-3}}(Fib_n) = rot_{F_{n-3}}(Fib_{n-3}Fib_{n-4}Fib_{n-3}Fib_{n-3}Fib_{n-4}) = Fib_{n-4}Fib_{n-3}Fib_{n-1}. \tag{3}$$

By Lemma 10(b) these strings are seeds of Fib^3_{n+1}, hence $\mathbf{CF}_{n+1}$ contains F_n at these positions.

(b) We prove by induction that $A = S_{n-2}F_nS_{n-3}$, $B = S_{n-2}F_n$, $C = S_{n-1}F_n$.

C is obtained from AB from the previous step by changing the last element from F_{n-1} to F_n. This AB is a prefix of $\mathbf{CF}_{n-1}$ of length $F_{n-2} = |S_{n-1}| + 1$.

B is obtained from C from the previous step which is obtained from AB for $n-2$ again by first changing F_{n-2} into F_{n-1} and then to F_n.

A is obtained from CB from the previous step; we again have the same construction. □

The recursive characterizations of Theorem 2 easily imply efficient algorithms for computing the **CF** array as well as its subsequent elements.

Theorem 3.

(a) $\mathbf{CF}_n$ can be computed in linear ($\mathcal{O}(F_n)$) time.
(b) Given n and k, we can compute $\mathbf{CF}_n[k]$ in $\mathcal{O}(n)$ time and $\mathcal{O}(1)$ space.

Proof. The first point follows directly from the previous theorem.

For a proof of (b), we use the following algorithm ComputeCF that implements the formula of Theorem 2(b). □

Corollary 2. *We have* $\mathbf{CCSet}(Fib_n) = \{F_3, \dots, F_n\}$. *More precisely, for $n \geq 4$, $\mathbf{CF}_n$ contains 4 occurrences of F_n, F_{n-3} occurrences of 3 and $2F_{n-k}$ occurrences of F_k for $3 < k < n$.*

Proof. From Theorem 2(b) we have that $S_n = S_{n-2}\,F_n\,S_{n-3}\,F_n\,S_{n-2}$. It is enough to prove by induction that S_n and S_{n-1} contain together F_{n-3} occurrences of 3 and $2F_{n-k}$ occurrences of F_k for $3 < k \leq n$. The base cases for $n = 4, 5$ are illustrated by Fig. 8. The content of $S_n + S_{n-1}$ is equal to the contents of $(S_{n-2} + S_{n-3} + S_{n-2}) + S_{n-1} = (S_{n-2} + S_{n-3}) + (S_{n-1} + S_{n-2})$ plus two occurrences of F_n. By induction for $n \geq 6$ we have $F_{n-5} + F_{n-4} = F_{n-3}$ occurrences of 3 and $2F_{n-k-2} + 2F_{n-k-1} = 2F_{n-k}$ occurrences of F_k for $3 < k \leq n-2$. F_{n-1} occurs $2F_1 = 2$ times (both in S_{n-1}). □

Algorithm 2: ComputeCF(n, k).

```
while $n \geq 4$ do
    if $k+1 \in \{F_{n-3}, F_{n-2}, F_{n-1}, F_n\}$ then
        return $F_n$
    if $k \geq F_{n-1}$ then $n := n-1;\ k := k - F_n$
    else if $k \geq F_{n-2}$ then $n := n-2;\ k := k - F_n$
    else if $k \geq F_{n-3}$ then $n := n-3;\ k := k - F_n$
    else $n := n-2$
return $F_n$
```

Let $repr_n(k)$ be a Fibonacci representation of $k+1$ where the most significant (leftmost) digit represents F_n. E.g., for $n = 10$ we have $49 + 1 = 50 = 34 + 13 + 3 = F_8 + F_6 + F_3$, hence $repr_{10}(49) = 0010100100$. Further let $odd(k) = k \bmod 2$ and $even(k) = 1 - odd(k)$.

Theorem 4. *Assume that $repr_n(k)$ ends with 0^x10^y, where $x, y \geq 0$ are maximal. Then $\mathbf{CF}_n[k] = F_l$, where*

$$l = \begin{cases} n & \text{if } x+y = n-1,\ x \leq 1 \\ odd(x) + y + 3 & \text{if } x+y = n-1, x > 1 \\ even(x) + y + 3 & \text{otherwise.} \end{cases}$$

Proof. Let us recall the algorithm of Theorem 3(b) and check what happens to $repr_n(k)$ in all the cases:

- If $repr_n(k) = 0^x10^y$ and $x \in \{0, 1\}$, then $k+1$ equals F_n or F_{n-1}, hence $l = n$.
- If $repr_n(k) = 0^x10^y$ and $x \in \{2, 3\}$, then $l = x \bmod 2 + y + 3 = n$ as well.
- If $repr_n(k) = 0^x10*$ and $x \in \{1, 2, 3\}$, then $repr_{n'}(k')$ (representation for the reduced n and k) will be equal to $00*$ (leading 0's are erased and 1 is changed into 0).
- If $repr_n(k) = 0^x10*$ and $x > 3$ then $repr_{n'}(k')$ will be equal to $0^{x-2}10*$ (n decreases by 2).

Hence for $repr_n(k) = 0^x10^y$ two trailing 0's will be erased unless $x \leq 3$, hence in the important step (in which output is produced) $repr_{n'}(k')$ will be equal to $0^{x \bmod 2+2}10^y$, hence n' will be equal to $x \bmod 2 + 2 + 1 + y = l$.

For $repr_n(k) = *10^x10^y$ in some step it will be changed into $0^{x+1}10^y$ (reduction to the previous case). □

Corollary 3. *We can output $\mathbf{CF}_n[i\,..\,j]$ in $\mathcal{O}(n + (j-i))$ time using $\mathcal{O}(\log(j-i))$ working space.*

Proof. In $\mathcal{O}(n)$ time we compute $repr_n(i)$ and store the positions of 1's in a sorted linked list starting from the least significant bit (e.g., for $i = 49$ the list contains $(3, 6, 8)$).

By Theorem 4 we can compute $\mathbf{CF}_n[k]$ in constant time using $repr_n(k)$ (the positions of two least significant 1's are stored at the beginning of the list, and this information is sufficient for our purpose).

We can update the list storing $repr_n(k)$ to obtain the list storing $repr_n(k+1)$ in constant amortized time (the possible extra $\mathcal{O}(n)$ time for small $(j-i)$ is covered in the complexity).

The above gives an algorithm which works in $\mathcal{O}(n)$ space. However, using a simple trick we can reduce it to $\mathcal{O}(\log(j-i))$. Consider the most significant bit on which $repr_n(i)$ and $repr_n(j)$ differ. It is enough to remember only one higher bit (all higher bits will be constant during the entire runtime). Let x be the number for which $repr_n(x)$ is equal to $repr_n(j)$ with all low bits zeroed (bits less important than the one which differs $repr_n(i)$ and $repr_n(j)$). From x to j we will only have up to $\Theta(\log(j-x))$ 1's (not counting the forgotten high bits). For the sequence from i to $x-1$ we count the new highest differing bit. This bit is $\Theta(\log(x-i))$ and $\max(\log(x-i), \log(j-x)) = \mathcal{O}(\log(j-i))$. □

6. Conclusions and open problems

Breslauer [3] proposed a linear-time algorithm for computing the shortest cover of every prefix of a string. We have proposed an $\mathcal{O}(n \log n)$-time algorithm for computing the shortest cover of every cyclic shift of a string. It remains an open problem if these values can be computed in $\mathcal{O}(n)$ time.

$\mathcal{O}(n)$, $\mathcal{O}(n \log n)$ and $\mathcal{O}(n^2)$-time algorithms for computing the shortest left seed, right seed, and seed, respectively, of all the prefixes of a string are known; see [25,26]. Here left and right seed are notions that are intermediate between cover and seed. It remains an open problem if the shortest left seed, right seed, and seed can be computed efficiently for all the cyclic shifts of a string.

Based on computer experiments we make the following conjecture.

Conjecture 1. *For a string S of length n, $|\mathbf{CCSet}(S)| = \mathcal{O}(\log n)$.*

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.

References

[1] M. Farach, Optimal suffix tree construction with large alphabets, in: 38th Annual Symposium on Foundations of Computer Science, FOCS 1997, IEEE Computer Society, 1997, pp. 137–143, https://doi.org/10.1109/SFCS.1997.646102.

[2] A. Apostolico, M. Farach, C.S. Iliopoulos, Optimal superprimitivity testing for strings, Inf. Process. Lett. 39 (1) (1991) 17–20, https://doi.org/10.1016/0020-0190(91)90056-N.

[3] D. Breslauer, An on-line string superprimitivity test, Inf. Process. Lett. 44 (6) (1992) 345–347, https://doi.org/10.1016/0020-0190(92)90111-8.

[4] D.W.G. Moore, W.F. Smyth, An optimal algorithm to compute all the covers of a string, Inf. Process. Lett. 50 (5) (1994) 239–246, https://doi.org/10.1016/0020-0190(94)00045-X.

[5] D.W.G. Moore, W.F. Smyth, A correction to "An optimal algorithm to compute all the covers of a string", Inf. Process. Lett. 54 (2) (1995) 101–103, https://doi.org/10.1016/0020-0190(94)00235-Q.

[6] Y. Li, W.F. Smyth, Computing the cover array in linear time, Algorithmica 32 (1) (2002) 95–106, https://doi.org/10.1007/s00453-001-0062-2.

[7] C.S. Iliopoulos, D.W.G. Moore, K. Park, Covering a string, Algorithmica 16 (3) (1996) 288–297, https://doi.org/10.1007/BF01955677.

[8] M. Christou, M. Crochemore, C.S. Iliopoulos, Quasiperiodicities in Fibonacci strings, Ars Comb. 129 (2016) 211–225.

[9] C.S. Iliopoulos, D.W.G. Moore, W.F. Smyth, The covers of a circular Fibonacci string, J. Comb. Math. Comb. Comput. 26 (1998) 227–236.

[10] T. Kociumaka, M. Kubica, J. Radoszewski, W. Rytter, T. Waleń, A linear time algorithm for seeds computation, in: Y. Rabani (Ed.), Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012, SIAM, 2012, pp. 1095–1112, https://doi.org/10.1137/1.9781611973099.86.

[11] T. Kociumaka, M. Kubica, J. Radoszewski, W. Rytter, T. Waleń, A linear-time algorithm for seeds computation, ACM Trans. Algorithms 16 (2) (2020) 27:1–27:23, https://doi.org/10.1145/3386369.

[12] M. Crochemore, C.S. Iliopoulos, J. Radoszewski, W. Rytter, J. Straszyński, T. Waleń, W. Zuba, Shortest covers of all cyclic shifts of a string, in: WALCOM: Algorithms and Computation - 14th International Conference, WALCOM 2020, in: Lecture Notes in Computer Science, vol. 12049, Springer, 2020, pp. 69–80, https://doi.org/10.1007/978-3-030-39881-1_7.

[13] R.M. Kolpakov, G. Kucherov, Finding maximal repetitions in a word in linear time, in: 40th Annual Symposium on Foundations of Computer Science, FOCS 1999, IEEE Computer Society, 1999, pp. 596–604, https://doi.org/10.1109/SFFCS.1999.814634.

[14] H. Bannai, T. I, S. Inenaga, Y. Nakashima, M. Takeda, K. Tsuruta, The "runs" theorem, SIAM J. Comput. 46 (5) (2017) 1501–1514, https://doi.org/10.1137/15M1011032.

[15] M. Crochemore, C.S. Iliopoulos, M. Kubica, J. Radoszewski, W. Rytter, T. Waleń, Extracting powers and periods in a word from its runs structure, Theor. Comput. Sci. 521 (2014) 29–41, https://doi.org/10.1016/j.tcs.2013.11.018.

[16] M. Crochemore, W. Rytter, Squares, cubes, and time-space efficient string searching, Algorithmica 13 (5) (1995) 405–425, https://doi.org/10.1007/BF01190846.

[17] H. Bannai, S. Inenaga, D. Köppl, Computing all distinct squares in linear time for integer alphabets, in: 28th Annual Symposium on Combinatorial Pattern Matching, CPM 2017, in: LIPIcs, vol. 78, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017, pp. 22:1–22:18, https://doi.org/10.4230/LIPIcs.CPM.2017.22.

[18] A. Deza, F. Franek, A. Thierry, How many double squares can a string contain?, Discrete Appl. Math. 180 (2015) 52–69, https://doi.org/10.1016/j.dam.2014.08.016.

[19] A.S. Fraenkel, J. Simpson, How many squares can a string contain?, J. Comb. Theory, Ser. A 82 (1) (1998) 112–120, https://doi.org/10.1006/jcta.1997.2843.

[20] D. Gusfield, J. Stoye, Linear time algorithms for finding and representing all the tandem repeats in a string, J. Comput. Syst. Sci. 69 (4) (2004) 525–546, https://doi.org/10.1016/j.jcss.2004.03.004.

[21] T. Kociumaka, J. Radoszewski, W. Rytter, T. Waleń, Internal pattern matching queries in a text and applications, in: P. Indyk (Ed.), Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, SIAM, 2015, pp. 532–551, https://doi.org/10.1137/1.9781611973730.36.

[22] A. Apostolico, F.P. Preparata, Data structures and algorithms for the string statistics problem, Algorithmica 15 (5) (1996) 481–494, https://doi.org/10.1007/BF01955046.

[23] P. Séébold, Propriétés combinatoires des mots infinis engendrés par certains morphismes, Report no. 85-16, LITP, Paris, 1985.

[24] C.S. Iliopoulos, D.W.G. Moore, W.F. Smyth, A characterization of the squares in a Fibonacci string, Theor. Comput. Sci. 172 (1–2) (1997) 281–291, https://doi.org/10.1016/S0304-3975(96)00141-7.

[25] M. Christou, M. Crochemore, O. Guth, C.S. Iliopoulos, S.P. Pissis, On left and right seeds of a string, J. Discret. Algorithms 17 (2012) 31–44, https://doi.org/10.1016/j.jda.2012.10.004.

[26] M. Christou, M. Crochemore, C.S. Iliopoulos, M. Kubica, S.P. Pissis, J. Radoszewski, W. Rytter, B. Szreder, T. Waleń, Efficient seed computation revisited, Theor. Comput. Sci. 483 (2013) 171–181, https://doi.org/10.1016/j.tcs.2011.12.078.