



Linear-Time Computation of Shortest Covers of All Rotations of a String

Maxime Crochemore  

King's College London, UK
Université Gustave Eiffel,
Marne-la-Vallée, France

Costas S. Iliopoulos  

King's College London, UK

Jakub Radoszewski  

University of Warsaw, Poland

Wojciech Rytter  


University of Warsaw, Poland

Juliusz Straszypiński  

University of Warsaw, Poland

Tomasz Waleń  

University of Warsaw, Poland

Wiktor Zuba  

University of Warsaw, Poland
CWI, Amsterdam, The Netherlands

Abstract

We show that lengths of shortest covers of all rotations of a length- n string over an integer alphabet can be computed in $\mathcal{O}(n)$ time in the word-RAM model, thus improving an $\mathcal{O}(n \log n)$ -time algorithm from Crochemore et al. (*Theor. Comput. Sci.*, 2021). Similarly as Crochemore et al., we use a relation of covers of rotations of a string S to seeds and squares in S^3 . The crucial parameter of a string S is the number $\xi(S)$ of primitive covers of all rotations of S . We show first that the time complexity of the algorithm from Crochemore et al. can be slightly improved which results in time complexity $\Theta(\xi(S))$. However, we also show that in the worst case $\xi(S)$ is $\Omega(|S| \log |S|)$. This is the main difficulty in obtaining a linear time algorithm. We overcome it and obtain yet another application of runs in strings.

2012 ACM Subject Classification Theory of computation \rightarrow Pattern matching

Keywords and phrases cover, quasiperiod, cyclic rotation, seed, run

Digital Object Identifier 10.4230/LIPIcs.CPM.2022.22

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

Juliusz Straszypiński: Supported by the Polish National Science Center, grant no. 2018/31/D/ST6/03991.

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

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

1 Introduction

A string C is a *cover* of a string S if each position in S is inside at least one occurrence of C in S . We say that a string Y is a *rotation* of a string X if $X = AB$ and $Y = BA$ for some strings A and B ; in this case we write $Y = \text{rot}_{|A|}(X)$, where $|A|$ is the length of string A .

Let us denote by $\text{CC}[i]$ the length of the shortest cover of $\text{rot}_i(S)$, where S is an input string. We consider the following problem.

COVERS OF ALL ROTATIONS

Input: A length- n string S .

Output: The array $\text{CC}[0..n-1]$.



© Maxime Crochemore, Costas S. Iliopoulos, Jakub Radoszewski, Wojciech Rytter, Juliusz Straszypiński, Tomasz Waleń, and Wiktor Zuba;

licensed under Creative Commons License CC-BY 4.0

33rd Annual Symposium on Combinatorial Pattern Matching (CPM 2022).

Editors: Hideo Bannai and Jan Holub; Article No. 22; pp. 22:1–22:15



Leibniz International Proceedings in Informatics

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

22:2 Linear-Time Computation of Shortest Covers of All Rotations of a String

Let Fib_n denote the n -th Fibonacci string ($Fib_0 = b$, $Fib_1 = a$, $Fib_n = Fib_{n-1}Fib_{n-2}$).

► **Example 1.** For the Fibonacci string $S = Fib_6$ we have

$$CC = [5, 5, 13, 3, 13, 5, 5, 13, 3, 8, 8, 3, 13]$$

The following table gives shortest covers for consecutive rotations $rot_0(S), \dots, rot_{12}(S)$.

i	rotation $rot_i(S)$	shortest cover	length $CC[i]$
0	abaababaabaab	abaab	5
1	baababaabaaba	baaba	5
2	aababaabaabab	aababaabaabab	13
3	ababaabaababa	aba	3
4	babaabaababaa	babaabaababaa	13
5	abaabaababaab	abaab	5
6	baabaababaaba	baaba	5
7	aabaababaabab	aabaababaabab	13
8	abaababaababa	aba	3
9	baababaababaa	baababaa	8
10	aababaababaab	aababaab	8
11	ababaababaaba	aba	3
12	babaabaababaa	babaabaababaa	13

Covers in strings are an extensively studied notion in stringology; algorithms computing covers in a string were proposed in [1, 4, 19, 18], not to mention approximate and generalized variants of covers (for a recent survey, see [10]). In [8] the authors showed an $\mathcal{O}(n \log n)$ -time (and $\mathcal{O}(n)$ space) algorithm that computes the lengths of shortest covers of all rotations of a length- n string. Later in [7] the authors developed a data structure over a length- n string that allows to answer queries about lengths of shortest covers of factors of the string. If combined with a data structure answering Weighted Ancestor Queries in the suffix tree in constant time after linear-time preprocessing, proposed in a very recent result [3] or in an off-line setting in [15], the data structure of [7] requires $\mathcal{O}(n \log n)$ -time and space preprocessing and allows to answer shortest cover queries for factors in $\mathcal{O}(\log n)$ time (amortized in the case that off-line Weighted Ancestor Queries are used). Hence, this result also yields an $\mathcal{O}(n \log n)$ -time algorithm for shortest covers of all rotations of a length- n string S if applied for all length- n factors of S^2 , despite being far more general. This suggests that perhaps shortest covers of all rotations can be computed in $o(n \log n)$ time. In comparison, shortest covers of all prefixes of a string can be computed in linear time using the on-line algorithm for computing shortest covers by Breslauer [4] (regardless of the alphabet size).

We show that this supposition is right by developing a linear-time algorithm computing shortest covers of all rotations of a given string.

Our algorithm works on a word-RAM model with word size $w = \Omega(\log n)$. We assume that the input string is over a so-called integer alphabet $[0..n^{\mathcal{O}(1)}]$, where n is the length of the string, which is a common assumption in the field (see, e.g., [11]).

Our approach. We use an approach based on runs and on packed representations of sets. We say that a string C is a seed of a string S if C is a cover of a superstring of S . In [8] it is shown that a string C is a cover of a rotation of a string S , $|C| \leq |S|$, if and only if C^2 is a factor of S^3 and C is a seed of S^3 . Each run in S^3 represents in a natural way the set of occurrences of primitively-rooted squares and we need to extract from these sets the occurrences of squares which are also seeds.

2 Preliminaries and algorithmic toolbox

We consider strings over an integer alphabet. Letters of a string S are numbered 0 through $|S| - 1$, with $S[i]$ being the i th letter. A factor of S is a string $S[i] \dots S[j]$, for any $0 \leq i \leq j < |S|$; it is denoted as $S[i..j]$ or $S[i..j+1)$. If $i > j$, we assume that $S[i..j]$ is the empty string. Throughout the paper, factors of S are represented in $\mathcal{O}(1)$ space by specifying the indices i, j of any of their occurrences $S[i..j]$. By $Occ(U, S)$ we denote the set of starting positions of occurrences of U in S . If $U = rot_i(V)$, we say that U and V are cyclically equivalent.

A string S has a period $p > 0$ if $S[i] = S[i + p]$ for all $i \in [0..|S| - p - 1]$. The smallest period of S is denoted as $\text{per}(S)$. A string S is called periodic if $2 \cdot \text{per}(S) \leq |S|$, and aperiodic otherwise. A string S is called primitive if $S = V^k$ for a positive integer k implies that $k = 1$.

2.1 Suffix tree

Let $ST(S)$ denote the suffix tree of string S . It can be constructed in linear time for a string over an integer alphabet [11].

The locus of a factor U of S is an (explicit or implicit) node of $ST(S)$ such that the path from the root to this node has string label U . An implicit node is represented by its nearest explicit descendant and its distance to the descendant. The string depth of an (explicit or implicit) node v is the length of the string label of v .

We use *Weighted Ancestor Queries* on a suffix tree. Such queries, given an explicit node v and an integer value ℓ that does not exceed the string depth of v , ask for the highest explicit ancestor u of v with string depth at least ℓ . We use the following very recent result.

► **Lemma 2** ([3]). *Let $ST(S)$ be the suffix tree of S . *Weighted Ancestor Queries* on $ST(S)$ can be answered in $\mathcal{O}(1)$ time after linear-time preprocessing.*

► **Corollary 3.** *After $\mathcal{O}(n)$ time preprocessing, the locus of any factor of a length- n string S can be computed in $\mathcal{O}(1)$ time.*

A simpler off-line version of *Weighted Ancestor Queries*, that would be sufficient for our purposes, with the same time guarantees was proposed earlier in [15]. We also use the following application of the queries.

► **Lemma 4.** *Any $\mathcal{O}(n)$ factors of a length- n string S can be ordered lexicographically in $\mathcal{O}(n)$ time.*

Proof. Assume that the explicit nodes of $ST(S)$ are numbered in pre-order. Let the locus of a factor be a pair (v, d) where v is the number of the explicit descendant and d is the distance; $d = 0$ for an explicit locus. Then it suffices to use Radix Sort to order the loci of the factors by non-decreasing first components, and by non-increasing second components in case of a tie. ◀

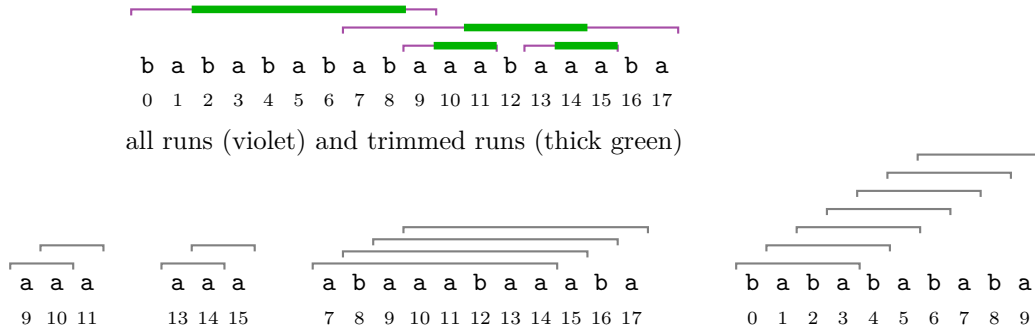
2.2 Runs, trimmed runs and p-squares

A string of the form $U^2 = UU$ is called a square. A square U^2 is called a p-square if U is primitive. We denote by $\frac{1}{2}\text{PSquares}(S)$ (p-square halves) the set of primitive factors Z of S such that the square Z^2 is also a factor of S .

The maximum number of occurrences of p-squares in a string of length n is $\Theta(n \log n)$ [9], whereas the total number of distinct square factors (hence, of distinct p-square factors) in a string is $\mathcal{O}(n)$ [13].

A *run* (also known as a *maximal repetition*) in S is a periodic fragment $R = S[i..j]$ which can be extended neither to the left nor to the right without increasing the period $p = \text{per}(R)$, i.e. if $i > 0$ then $S[i - 1] \neq S[i + p - 1]$ and if $j < |S| - 1$ then $S[j + 1] \neq S[j - p + 1]$. The exponent of a run R , denoted as $\text{exp}(R)$, is defined as $(j - i + 1)/p$. Let $\mathcal{R}(S)$ denote the set of all runs of string S . For a length- n string S it holds that $|\mathcal{R}(S)| = \mathcal{O}(n)$; moreover, the sum of exponents of runs in S is $\mathcal{O}(n)$ [17].

The center of an occurrence $S[a..a + 2\ell)$ of a square U^2 is defined as the position $a + \ell$. A square occurrence $S[a..b]$ is said to be *induced* by a run $R = S[i..j]$ if $i \leq a, b \leq j$ and $\text{per}(R) = \text{per}(U)$. Every square is induced by exactly one run [6]. A run $R = S[i..j]$ with period p induces p -squares with centers at positions in $[i + p..j - p + 1]$; the length of this interval is $|R| - 2 \cdot \text{per}(R) + 1$.



■ **Figure 1** Four runs (presented at the top) generate all p -squares (bottom). The total length of trimmed runs is the same as the number of occurrences of p -square factors of S .

For a run $R = S[i..j]$ with period p , we define a *trimmed run* as $S[i + p..j - p + 1]$. We assume that the trimmed run stores the period of the original run. We denote by $\mathcal{R}'(S)$ the set of trimmed runs of S . The following lemma was already shown in [5]; here we give a more direct proof in terms of p -squares.

► **Lemma 5.** *For any string S of length n we have that $\sum_{R \in \mathcal{R}'(S)} |R| = \mathcal{O}(n \log n)$.*

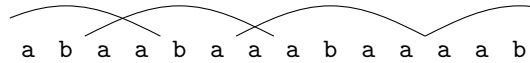
Proof. The run with period p corresponding to a trimmed run R induces $|R|$ occurrences of p -squares with half length p , whose centers are positions of R , and vice versa. Now the thesis follows directly from the fact that the number of occurrences of primitively rooted squares is $\mathcal{O}(n \log n)$. ◀

The *Lyndon root* of a run R is the lexicographically smallest rotation of its length- $\text{per}(R)$ prefix. If L is the Lyndon root of a run R , then R can be uniquely represented as (L, y, a, b) for $0 \leq a, b < |L|$ such that $R = L[|L| - a..|L| - 1]L^yL[0..b]$; we call this the *Lyndon representation* of R . One can group all runs in S by Lyndon roots and compute the Lyndon representations of all runs in $\mathcal{O}(n)$ time; see [6].

The *Lyndon type* of a primitive string U , denoted as $\text{LynType}(U)$, is the lexicographically smallest rotation of U .

2.3 Relation to seeds

A factor C of a string S is called a *seed* of S if there exists a string S' having S as a factor such that C is a cover of S' . In other words, C is a seed of S if S is covered by occurrences and left and right overhangs of C ; see Figure 2.



■ **Figure 2** A string with a seed aaba.

► **Lemma 6** ([15]). *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 tree of the string. This representation can be computed in $\mathcal{O}(n)$ time.*

Weighted Ancestor Queries together with a representation of all the seeds of a string from Lemma 6 can be used to show the following lemma.

► **Lemma 7** ([8, see Lemma 8]). *Let S be a string of length n . Given a family \mathcal{U} of $\mathcal{O}(n)$ factors of S , all strings in \mathcal{U} that are seeds of S can be reported in $\mathcal{O}(n)$ time.*

By $\mathbf{PrimCov}[i]$ we denote the set of lengths of covers of $rot_i(S)$ that are primitive strings. Let us observe that each of these sets is non-empty.

► **Observation 8.** $\mathbf{CC}[i] = \min \mathbf{PrimCov}[i]$.

Proof. Every string U has a cover, possibly equal to U . The shortest cover C of every string U is primitive. Otherwise, if we had $C = D^k$ for integer $k > 1$, then D would be a shorter cover of U . ◀

We further denote by $\mathbf{Seeds}(S)$ the set of factors which are seeds of S . The following lemma uses these sets for the string $X := S^3$ in order to characterize covers of all rotations of S . Denote by $\mathbf{Centers}(C^2, X)$ the centers of all occurrences of C^2 in X .

► **Lemma 9** ([8, Lemma 3]). [**Covers = Seeds+Squares**]
Let S be a string of length n , $X = S^3$, and C be a string of length up to n . Then $|C| \in \mathbf{PrimCov}[i]$ if and only if $C \in \mathbf{Seeds}(X) \cap \frac{1}{2}\mathbf{PSquares}(X)$ and $n + i \in \mathbf{Centers}(C^2, X)$.

We denote $\xi(S) = \sum_{i=0}^{n-1} |\mathbf{PrimCov}[i]|$.

3 A version of the algorithm in [8]

The algorithm from [8] computes the array \mathbf{CC} in $\mathcal{O}(n \log n)$ time and is based on the characterization of Lemma 9. For each p-square Z^2 in X such that $|Z| \leq n$, if Z is a seed of X , then for every $j \in \mathit{Occ}(Z^2, X)$, we set $\mathbf{CC}[(j + |Z|) \bmod n]$ to the minimum of its current value (starting from n) and $|Z|$; see Algorithm 1.

■ **Algorithm 1** Computing \mathbf{CC} array as in [8].

```

1  $X := S^3$ ;  $\mathbf{CC}[0..n] = (n, \dots, n)$ 
2 foreach  $Z \in \frac{1}{2}\mathbf{PSquares}(X) \cap \mathbf{Seeds}(X)$  do
3   foreach  $j \in \mathit{Occ}(Z^2, X)$  do
4      $i := (j + |Z|) \bmod n$ 
5      $\mathbf{CC}[i] := \min(\mathbf{CC}[i], |Z|)$ 

```

Lemma 9 directly implies the following observation.

► **Observation 10.** *For each execution of line 5 in Algorithm 1, we have $|Z| \in \mathbf{PrimCov}[i]$.*

In the following lemma we improve the worst case complexity analysis of Algorithm 1. The proof of the lemma generally follows the details of the algorithm from [8].

► **Lemma 11.** *The time complexity of Algorithm 1 is $\Theta(\xi(S))$.*

Proof. All distinct p-squares in X can be computed in $\mathcal{O}(n)$ time using the algorithm from [6]. Lemma 7 can be used to check which of the p-square halves are seeds of X . For each p-square Z^2 whose half satisfies this condition, we find its locus v in $ST(X)$ using Corollary 3. Up to this point, all the steps work in $\mathcal{O}(n)$ time. Finally, all $j \in Occ(Z^2, X)$ can be listed in time proportional to the number of these elements by inspecting all leaves in the subtree of v in $ST(X)$. For each j we perform the instruction in line 5; by Observation 10, it corresponds to a (distinct) element from $\mathbf{PrimCov}[i]$. Overall the time complexity is $\Theta(n + \xi(S))$; however, $\xi(S) \geq n$ by Observation 8. ◀

A proof of the following theorem is deferred until Section 6. The theorem implies that, even with the improved complexity analysis, the algorithm from [8] works in $\Omega(n \log n)$ time in the worst case.

► **Theorem 12.** *There exist infinitely many strings S such that $\xi(S) = \Omega(|S| \log |S|)$.*

► **Remark 13.** The algorithm from Section 3 can be easily modified with the aid of internal Two-Period Queries of [16, 2] (such a query computes the smallest period of a factor of the text in case that the factor is periodic) to output only aperiodic covers of rotations of a string in time proportional to their total number. Still, the construction of Theorem 12 actually provides $\Omega(|S| \log |S|)$ aperiodic covers of rotations of a string.

In our approach we heavily use runs. Hence the next version of the algorithm is based on runs and is more suitable for further improvements. We also substitute the formula with the modulo operation by a formula that closely follows Lemma 9.

■ **Algorithm 2** A version of Algorithm 1 employing runs.

```

1  $X := S^3$ ;  $\mathbf{CC}[0..n] = (n, \dots, n)$ 
2 foreach trimmed run  $X[a..b]$  in  $X$  with period  $p$  do
3   for  $i := a$  to  $b$  do
4     if  $i \in [n..2n]$  and  $X[i..i+p] \in \mathbf{Seeds}(X)$  then
5        $\mathbf{CC}[i-n] := \min(\mathbf{CC}[i-n], p)$ 

```

4 Two useful representations of p-squares: occurrences and values

Our problem reduces to finding for each position i the length of the shortest p-square centered at i , whose half is a seed of the string.

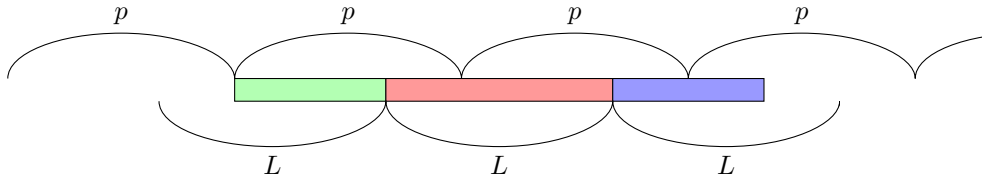
The trimmed runs, accompanied by additional useful data, can be treated as package representations of p-squares. Each occurrence of a p-square can be identified with a pair (i, p) , where i is the center and p is the period of the square.

► **Definition 14.** *An occurrence package $\gamma = (I, p)$ of occurrences of p-squares (occ-package, in short) corresponds to an interval I of consecutive centers of cyclically equivalent p-squares with period p such that $|I| \leq p$.*

With each occ-package γ we keep the following data

- $L = \mathit{LynType}(\gamma)$: the (common) Lyndon type of all the corresponding half-squares;
- $\mathit{first}(\gamma) = j_1$ and $\mathit{last}(\gamma) = j_2$, where $\mathit{rot}_{j_1}(L)$ and $\mathit{rot}_{j_2}(L)$ are the halves of the first and last p-square in γ .

An occ-package γ is called *canonical* if $first(\gamma) \leq last(\gamma)$. In this case the p-squares generated by γ are $rot_j(L)^2$ for $first(\gamma) \leq j \leq last(\gamma)$ and $L = LynType(\gamma)$. Our algorithms will only use canonical occ-packages.



■ **Figure 3** A trimmed run with period p (colored rectangles) represents the set of occurrences of p-squares with half length p that are induced by the corresponding run (top). In this example this set of occurrences is split into three (canonical) occ-packages. L is the Lyndon root of the run. A single run R generates at most $exp(R)$ occ-packages.

An *occurrence-representation* of occurrences of p-squares in X consists of a set of occ-packages for X containing all p-squares of X (together with all parameters defined above).

The lemma below follows from the fact that all runs can be computed in linear time, together with their Lyndon roots; see Figure 3.

► **Lemma 15.** *For any string X we can compute in linear time, using the runs of X , an occurrence-representation $\Gamma(X)$ of occurrences of all p-square factors of X .*

Proof. We compute the Lyndon representations of all runs in X [6]. For a run $R = X[i..j]$ with period p and Lyndon representation (L, y, a, b) , we form y occ-packages with intervals $[i + p..i + p + a)$, $[i + p + a..i + 2p + a)$, \dots , $[i + (y - 2)p + a..i + (y - 1)p + a)$, $[i + (y - 1)p + a..i + (y - 1)p + a + b]$ if $y \geq 2$, and a single package $[i + p..i + a + b]$ if $y = 1$. ◀

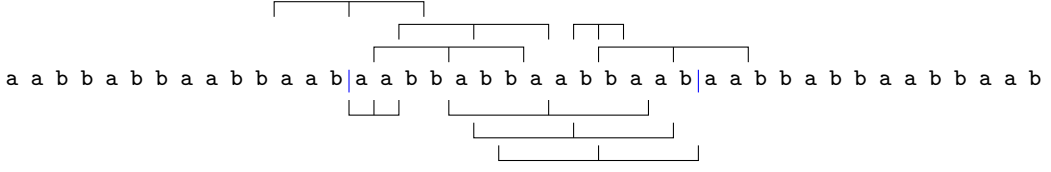
Let us note that the size of $\Gamma(X)$ is $\mathcal{O}(n)$ if $|X| = \mathcal{O}(n)$, even though the total length of intervals in occ-packages can be $\Omega(n \log n)$.

$\Gamma(X)$ is an interval representation of all occurrences of p-squares. We also need an interval-type representation of all distinct p-squares in X ; this time the sum of lengths will have linear total length.

Let a p-square U^2 be formally identified with (L, s) , where $U = rot_s(L)$ and L is the Lyndon root of U . Let us assume that all p-square factors of length up to $2n$ in X are ordered with respect to their (L, s) pairs. Let $HalfSquares[i]$ be the i -th p-square half U in this order. The length of the table $HalfSquares$ is $\mathcal{O}(n)$ [13].

► **Example 16.** For the string $S = aabbabbaabbaab$ and $X = S^3$, the table $HalfSquares$ looks as follows; see also Figure 4.

i	$HalfSquares[i]$	L	s	i	$HalfSquares[i]$	L	s
1	a	a	0	18	abba	aabb	1
2	aab	aab	0	19	bbaa	aabb	2
3	baa	aab	2	20	baab	aabb	3
4	aabaabbabbaabb	aabaabbabbaabb	0	21	abb	abb	0
...	22	bba	abb	1
17	baabaabbabbaab	aabaabbabbaabb	13	23	b	b	0



■ **Figure 4** All distinct p-squares of half length smaller than $|S|$ in $X = S^3$ for $S = \text{aabbabbaabbaab}$. For each of them, an example occurrence with the center in the middle S in X is shown.

We denote by *SeedMask* the Boolean vector such that

$$\text{SeedMask}[i] = 1 \Leftrightarrow \text{HalfSquares}[i] \in \mathbf{Seeds}(X).$$

In other words *SeedMask* is the characteristic vector of the set $\mathbf{Seeds}(X)$ as a subset of the set $\frac{1}{2}\mathbf{PSquares}(X)$.

Let us assume that an occ-package $\gamma = ([a..b], p)$ represents consecutive p-squares with halves U_1, U_2, \dots, U_k . We introduce a Boolean vector *SeedMask* $_{\gamma}$ such that *SeedMask* $_{\gamma}[i] = 1$ if and only if U_i is a seed of X .

- **Lemma 17.** (a) *The table SeedMask can be computed in $\mathcal{O}(n)$ time.*
 (b) *We can compute in $\mathcal{O}(n)$ time for all $\gamma = ([a..b], p) \in \Gamma(X)$ the values $\Phi(\gamma) := (a', b')$ such that $\text{SeedMask}_{\gamma}[a..b] = \text{SeedMask}[a'..b']$.*

Proof. (a) The main part is to compute the table *HalfSquares*. This is done as shown below in Algorithm 3.

■ **Algorithm 3** Compute *HalfSquares*.

```

1   $\mathcal{L} :=$  list of all occ-packages  $\gamma \in \Gamma(X)$  ordered by  $(\text{LynType}(\gamma), \text{first}(\gamma), -\text{last}(\gamma))$ 
2  foreach occ-package  $\gamma$  in  $\mathcal{L}$  but the first one do
3       $\gamma' :=$  previous occ-package in  $\mathcal{L}$ 
4      if  $\text{LynType}(\gamma') = \text{LynType}(\gamma)$  and  $[\text{first}(\gamma').. \text{last}(\gamma')] \supseteq [\text{first}(\gamma).. \text{last}(\gamma)]$  then
5          Remove  $\gamma$  from  $\mathcal{L}$ 
6  foreach occ-package  $\gamma$  in  $\mathcal{L}$  do
7      if there is a next occ-package  $\gamma'$  in  $\mathcal{L}$  and  $\text{LynType}(\gamma') = \text{LynType}(\gamma)$  then
8           $\text{end} := \min(\text{first}(\gamma') - 1, \text{last}(\gamma))$ 
9      else
10          $\text{end} := \text{last}(\gamma)$ 
11     Let  $\gamma = ([a..b], p)$ 
12     for  $i := a$  to  $a + \text{end} - \text{first}(\gamma)$  do
13         Append  $X[i..i + p]$  to HalfSquares
    
```

In the algorithm we use Lemma 15 to compute in $\mathcal{O}(n)$ time an occurrence representation $\Gamma(X)$ of occurrences of p-squares in X . We can sort all occ-packages $\gamma \in \Gamma(X)$ with respect to $(\text{LynType}(\gamma), \text{first}(\gamma), -\text{last}(\gamma))$ using Lemma 4 and then Radix Sort. Intuitively, the intervals $[\text{first}(\gamma).. \text{last}(\gamma)]$ for packages with equal Lyndon type are sorted from left to right, and intervals with equal start point are ordered by non-increasing end points. Then we scan the sorted list from left to right, removing redundant intervals $[\text{first}(\gamma).. \text{last}(\gamma)]$, that is, intervals that are contained in another such interval corresponding to the same Lyndon type. Finally, each of the non-redundant occ-packages generates some number of consecutive elements of *HalfSquares* list that were not generated by any previous occ-package in \mathcal{L} .

Let us recall that $SeedMask[i] = 1$ if and only if $HalfSquares[i]$ is a seed of X . Consequently the whole table $SeedMask$ can be computed in $\mathcal{O}(n)$ time due to Lemma 7.

(b) The function Φ can be inferred from the construction of the table $HalfSquares$. For each occ-package γ that remained on the list \mathcal{L} until the end, we set the first component of $\Phi(\gamma)$ to the position in $HalfSquares$ of the first half p-square that was introduced due to γ in line 13. For each remaining package γ , if γ' is the package that was used to remove γ from \mathcal{L} in line 4 and $\Phi(\gamma') = (a', b')$, then the first component of $\Phi(\gamma)$ is $a + first(\gamma) - first(\gamma')$. In either case the second component of $\Phi(\gamma)$ can be calculated from the first component and the length of the interval I in $\gamma = (I, p)$. ◀

Thus the bit-mask of each occ-package is a copy of a fragment of a single Boolean vector of length $\mathcal{O}(n)$; see also Figure 5.

Intuition. We need to know, in the representation $\Gamma(X)$, for each interval, a Boolean vector which says which half p-square is a seed. If we do it directly, this needs $\Omega(n \log n)$ space. However, these Boolean vectors are parts of the representation $SeedMask(X)$. Hence we can have a reference to a part of $SeedMask(X)$. The total number of references is asymptotically the same as the sum of exponents of runs, which is known to be linear. However, we need to pack $\mathcal{O}(\log n)$ -sized chunks of Boolean vectors into machine words. With further bit-level optimizations this eventually results in a linear-time algorithm. The exact implementation is given in Section 5, but first we provide an $\mathcal{O}(n \log n)$ -time implementation to illustrate the main ideas of our approach.

We introduce bitmasks $T_p[n..2n]$ for $p \in [1..n]$ such that $T_p[n+i] = 1$ if and only if $rot_i(S)$ has a cover of length at most p . We have

$$CC[i] = \min \{ p : T_p[n+i] = 1 \}.$$

For two equal-length bitmasks $F_1[a..a+\ell]$, $F_2[a'..a'+\ell]$ we define

$$\Delta(F_1, F_2) = \{ j \in [0.. \ell] : F_1[a+j] < F_2[a'+j] \}.$$

Using the bitmask tables T and $SeedMask$ we can write the next version of the algorithm. The variable New is the set of centers of new p-squares, whose halves are seeds of X (which is stored in the fragment of $SeedMask$ as a Boolean vector). In the actual implementation the bitmask T is extended to the range $[0..3n]$; this will be more convenient in the next version of the algorithm.

■ **Algorithm 4** Extraction of shortest covers of rotations.

```

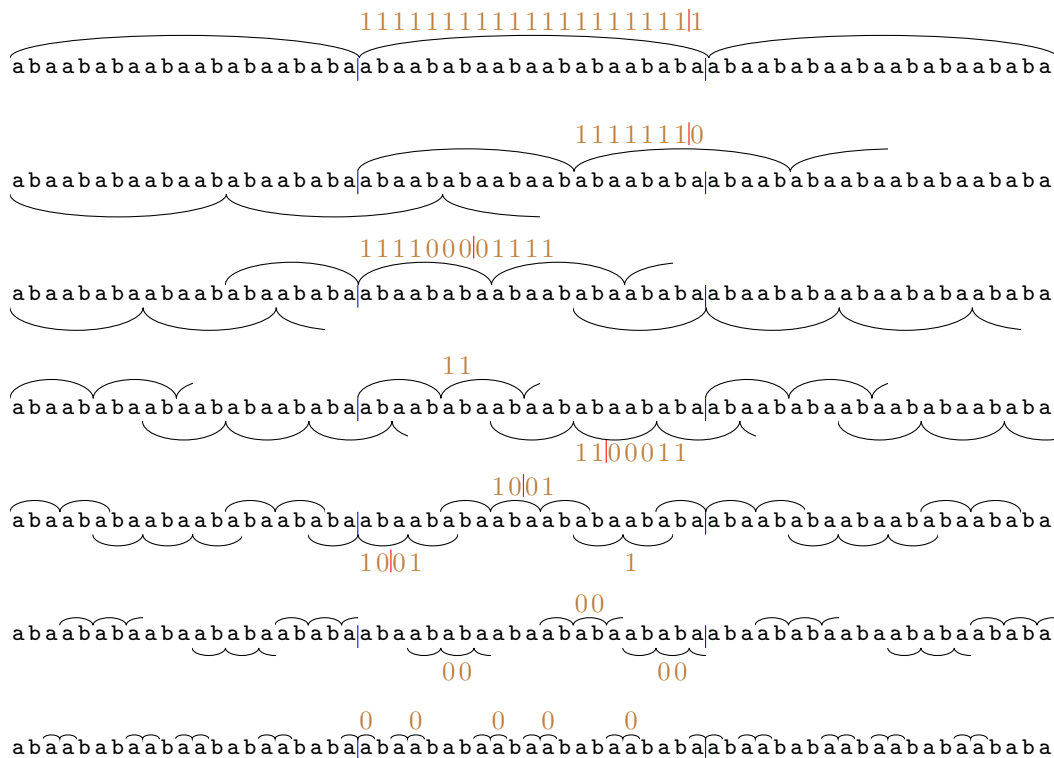
1 Compute  $\Gamma(X)$ ,  $SeedMask$  and  $\Phi$  function for  $X = S^3$ 
2  $T[0..3n] := (0, \dots, 0)$ 
3 for  $p := 1$  to  $n$  do  $\triangleright$  Invariant:  $T = T_{p-1}$ 
4   foreach occ-package  $\gamma = ([a..b], p)$  in  $\Gamma(X)$  do
5      $New := \Delta(T[a..b], SeedMask[a'..b'])$ 
6     foreach  $i \in New$  do
7       if  $a+i \in [n..2n]$  then  $CC[a+i-n] := p$ 
8        $T[a+i] := 1$ 

```

Algorithm 4 still has $\mathcal{O}(n \log n)$ time complexity since the total size of all processed fragments can be $\Omega(n \log n)$. However we process only a linear number of fragments. If we implement the assignment $T[i] := 1$ in constant time, and the operation Δ in time

22:10 Linear-Time Computation of Shortest Covers of All Rotations of a String

proportional to the size of its output (plus a constant), then the whole complexity will be linear. We achieve that in the next section by packing $\log n$ -sized chunks of Boolean vectors into single machine words.



■ **Figure 5** All runs of period at most $|S|$ in $X = S^3$ for $S = \text{Fib}_7$ are shown, grouped by their periods. For each run, the corresponding trimmed run generates one or more occ-packages γ (cf. Lemma 15). Each occ-package corresponds to a fragment of *SeedMask* (see Lemma 17); these fragments are shown limited to the positions in the middle occurrence of S (red lines are the delimiters between occ-packages and correspond to Lyndon types). For each position $i \in [n..2n]$, the ones in bitmask at position i correspond to primitive covers of $\text{rot}_i(S)$ (cf. Figure 6). The length of *SeedMask* is equal to the number of distinct p-square factors of half length at most n of X , i.e. $1+2+3+5+8+7+21 = 47$. For Fibonacci strings the total length of all trimmed runs is superlinear (as the number of **occurrences** of p-squares), while the length of *SeedMask* is always $\mathcal{O}(n)$ (as the number of distinct **values** of p-squares). This property is crucial to achieve a linear time algorithm.

5 Speeding up Algorithm 4

Instead of the original version of *SeedMask* and T we use now their packed versions. First we describe several operations on a $\log n$ -sized chunk of a Boolean vector packed into a single integer.

Bit-wise operations. Let $w = \Omega(\log n)$ denote the length of the machine word. We use the bitwise operations **and**, **not** from the word-RAM model as well as **ffs** (find first (bit) set; also known under the name **ctz** – count trailing zeroes), which computes the index of the

least significant set bit in a machine word. If `ffs` is not supported by the model, we can set the chunk length to $\frac{1}{2} \log n$ and preprocess the `ffs` values for each possible machine word in $\mathcal{O}(\sqrt{n} \log n)$ time.

We redefine the operation Δ to work on two machine words representing Boolean vectors in time proportional to the size of output plus $\mathcal{O}(1)$. To this end we apply the operation `ffs` as shown in following function (Algorithm 5).

■ **Algorithm 5** Bitmask realisation of $\Delta(F_1, F_2)$.

```

1  $F := \text{and}(\text{not}(F_1), F_2)$ 
2  $R := \emptyset$ 
3 while  $F \neq 0$  do
4    $i := \text{ffs}(F)$ 
5   Set  $i$ th bit of  $F$  to 0
6    $R := R \cup \{i\}$ 
7 return  $R$ 

```

We also use an operation $\text{Extract}(B, a, b, s)$ that, given a packed bitmask B , indices $0 \leq a \leq b < |B|$ and shift value $0 \leq s < w$, returns the fragment consisting of bits $B[a], \dots, B[b]$ also represented as a packed bitmask but shifted by s bits, i.e. the packed representation of the bitmask $0^s B[a..b] 0^t$, where $t = w - (b - a + 1 + s) \bmod w$. It can be implemented in $\mathcal{O}((b - a + 1)/\log n + 1)$ time on the word-RAM.

In the algorithm $\text{Packed}T$ and PackedSeed are packed representations, using $\mathcal{O}(n/\log n)$ machine words, of T and SeedMask .

■ **Algorithm 6** Packed bitmask realisation of Algorithm 4.

```

▷  $w = \Omega(\log n)$  is the length of machine word
1 Compute  $\Gamma(X)$ ,  $\text{SeedMask}$  and  $\Phi$  function for  $X = S^3$ 
2  $\text{Packed}T[0.. \lceil 3n/w \rceil - 1] := (0, \dots, 0)$ 
3 for  $p := 1$  to  $n$  do
4   foreach occ-package  $\gamma = ([a..b], p)$  do
5      $(a', b') := \Phi(\gamma)$ 
6      $F := \text{Extract}(\text{PackedSeed}, a', b', a \bmod w)$       ▷  $|F| = \mathcal{O}((b - a)/\log n + 1)$ 
7      $d := a \text{ div } w$ 
8     for  $j := 0$  to  $|F| - 1$  do
9        $\text{New} := \Delta(F[j], \text{Packed}T[j + d])$ 
10      foreach  $k \in \text{New}$  do
11         $i := (j + d) \cdot w + k$ 
12        if  $i \in [n..2n)$  then  $\text{CC}[i - n] := p$ 
13        Set  $k$ -th bit in  $\text{Packed}T[j + d]$  to 1

```

Thus we obtain the main result of this paper.

► **Theorem 18.** *The lengths of shortest covers of all rotations of a string can be computed in $\mathcal{O}(n)$ time.*

Proof. We apply Algorithm 6. The initial computations in line 1 are performed in $\mathcal{O}(n)$ time by Lemma 15 (computation of the occurrence-representation $\Gamma(X)$) and Lemma 17 (computation of the global SeedMask and the Φ function on occ-packages).

22:12 Linear-Time Computation of Shortest Covers of All Rotations of a String

By Lemma 5, the total size of $SeedMask_\gamma$ over all occ-packages γ is $\mathcal{O}(n \log n)$, and by Lemma 15, the number of occ-packages is $\mathcal{O}(n)$. The for-loop in line 8 iterates over the packed representations of seed masks, hence performs $\mathcal{O}(n)$ iterations in the course of the algorithm. The complexity of all calls to $Extract$ is the same.

The total number of iterations of the foreach-loop in line 10, equal to the total size of the sets New , is $\mathcal{O}(n)$, as each of them sets one new bit of $PackedT$ to 1. The set New is computed in $\mathcal{O}(|New| + 1)$ time as shown in Algorithm 5. The complexity follows. ◀

6 Proof of Theorem 12

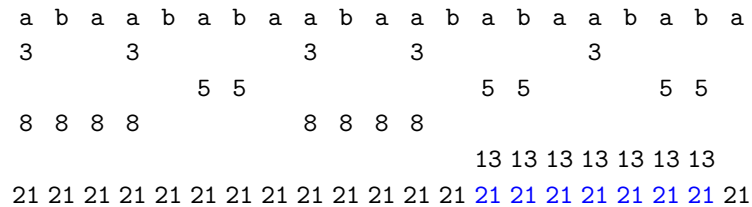
Let $F_m = |Fib_m|$ denote the m -th Fibonacci number. We start with a simple fact that is probably folklore.

► **Fact 19.** $rot_k(Fib_m)$ is aperiodic for $0 \leq k < F_{m-1}$.

Proof. A proof by induction. It is easy to check that the longest common prefix of strings Fib_m^2 and Fib_{m-1}^3 has length $F_{m+1} - 2$, hence F_{m-1} is a period of any such rotation (not small enough to make the string periodic). If such a rotation was periodic with a period $p \leq \frac{F_m}{2}$, then there would exist a square with half length p in this rotation, hence p would be a Fibonacci number [14] and $p \leq F_{m-2}$. Then by Fine and Wilf's periodicity lemma [12] the rotation would also have a period not exceeding $\gcd(F_{m-1}, p) \leq \frac{F_{m-1}}{2}$. However, $rot_k(Fib_m)$ for $0 \leq k < F_{m-2}$ contains as a prefix $rot_k(Fib_{m-1})$ and for $F_{m-2} \leq k < F_{m-1} - 1$ it contains a factor Fib_{m-1} (as $Fib_{m-2}^2 = Fib_{m-1}Fib_{m-4}$); by the inductive hypothesis those rotations are aperiodic. ◀

We also use the following fact that was already presented in [8].

► **Fact 20** (see [14, 8]). If $m \geq 3$, only strings of length F_k , for $3 \leq k \leq m$, can be covers of rotations of Fib_m .



■ **Figure 6** The figure shows the lists of lengths of primitive covers of rotations of Fib_7 . The numbers below the position correspond to the rotation beginning in that position. Apart from F_m , the numbers F_k form F_{m-k} blocks of $F_{k-1} - 1$ elements each. Numbers in blue correspond to primitive covers that are not aperiodic.

► **Lemma 21.** For $S = Fib_m$, we have $\xi(S) = \Theta(mF_m)$. The same bound holds if we count the aperiodic covers or all the covers of rotations of S .

Proof. We prove the lemma using a slight generalisation of the technique used in [8] to show a representation of lengths of shortest covers of all rotations of Fib_m .

Fact 20 shows that lengths of covers of any rotation belong to $\{F_3, \dots, F_m\}$, hence also that $\xi(S) = \mathcal{O}(mF_m)$ (the bound holds also for all covers of all rotations), thus we focus only on strings of those lengths when proving the lower bound.

In [8] it was shown that the lengths of shortest covers of rotations of Fib_m can be expressed in a concise way in relation to the shortest covers of rotations of Fib_k for $k < m$. More precisely, if S_m is the prefix of $\mathbf{CC}(Fib_m)$ of length $F_{m-1} - 1$, then [8, Theorem 2(b)] shows that for $m \geq 4$,

$$\mathbf{CC}(Fib_m) = S_{m-2}, F_m, S_{m-3}, F_m, S_{m-2}, F_m, S_{m-1}, F_m.$$

The proof of the theorem, however, never used the property that the covers were shortest, but only a division into covers of length F_{m-1} and the shorter ones.

Let A_m be a table of length $F_{m-1} - 1$ of lists such that $A_m[i]$ consists of all lengths of aperiodic covers of $rot_i(Fib_m)$. Since we only care about asymptotics, we will limit the proof to the first $F_{m-1} - 1$ rotations of Fib_m . We claim that $A_m = A'_{m-2}\{F_m\}A'_{m-3}\{F_m\}A'_{m-2}$, where A'_k equals A_k with every list appended with the value F_m .

Thanks to this limitation we do not need to care about covers of lengths F_{m-1} (they are not present for those rotations), which can behave differently than F_k for $k < m - 1$ in the recurrence. The values F_k for $k < m - 1$ behave according to the described recursive formula as shown in the proof of [8, Theorem 2(b)], while values F_m occur in each list since the whole rotations are their own covers and are aperiodic due to Fact 19.

Let C_m be the number of elements in all the lists in A_m . Then the definition of A_m provides a recursive formula $C_m = 2 \cdot C_{m-2} + C_{m-3} + F_{m-1} - 1$. From here it can be checked by simple induction that $C_m \geq cmF_m$ for any $c < \frac{1}{2+\phi}$ (where ϕ is the golden ratio) and sufficiently large m :

$$\begin{aligned} 2c(m-2)F_{m-2} + c(m-3)F_{m-3} + F_{m-1} - 1 &\geq cmF_m \Leftrightarrow \\ c(mF_m - (2m-4)F_{m-2} - (m-3)F_{m-3}) &\leq F_{m-1} - 1 \Leftrightarrow \\ c(mF_{m-1} - (m-4)F_{m-2} - (m-3)F_{m-3}) &\leq F_{m-1} - 1 \Leftrightarrow \\ c(4F_{m-2} + 3F_{m-3}) &\leq F_{m-1} - 1 \Leftrightarrow \\ c(F_m + 2F_{m-1}) &\leq F_{m-1} - 1 \Leftrightarrow \\ c &\leq \frac{F_{m-1} - 1}{F_m + 2F_{m-1}} \sim \frac{1}{2 + \phi} \end{aligned}$$

This concludes the proof. ◀

Proof of Theorem 12. By Lemma 21 for the family of Fibonacci strings we have $\xi(Fib_m) = \Omega(F_m \cdot m) = \Omega(|Fib_m| \log |Fib_m|)$. ◀

7 Final Remarks

In our algorithm we extract fragments consisting of $\mathcal{O}(n \log n)$ bits in total of a packed bitmask consisting of $\mathcal{O}(n)$ bits. The fact that all these fragments can be represented in $\mathcal{O}(n)$ machine words allows us to obtain linear time complexity. Each of these bitmask fragments carries the information about which subsequent occurrences of p-squares of the same half length are seeds of the string S^3 . Based on combinatorial properties of squares and seeds, it can be the case that the total size of RLE representations of these bitmask fragments is $\mathcal{O}(n)$. This would simplify the algorithm.

References

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

- 2 Hideo Bannai, Tomohiro I, Shunsuke Inenaga, Yuto Nakashima, Masayuki Takeda, and Kazuya Tsuruta. The “runs” theorem. *SIAM Journal on Computing*, 46(5):1501–1514, 2017. doi:10.1137/15M1011032.
- 3 Djamal Belazzougui, Dmitry Kosolobov, Simon J. Puglisi, and Rajeev Raman. Weighted ancestors in suffix trees revisited. In Pawel Gawrychowski and Tatiana Starikovskaya, editors, *32nd Annual Symposium on Combinatorial Pattern Matching, CPM 2021, July 5-7, 2021, Wrocław, Poland*, volume 191 of *LIPICs*, pages 8:1–8:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICs.CPM.2021.8.
- 4 Dany Breslauer. An on-line string superprimitivity test. *Information Processing Letters*, 44(6):345–347, 1992. doi:10.1016/0020-0190(92)90111-8.
- 5 Panagiotis Charalampopoulos, Jakub Radoszewski, Wojciech Rytter, Tomasz Waleń, and Wiktor Zuba. The number of repetitions in 2D-strings. In Fabrizio Grandoni, Grzegorz Herman, and Peter Sanders, editors, *28th Annual European Symposium on Algorithms, ESA 2020, September 7-9, 2020, Pisa, Italy (Virtual Conference)*, volume 173 of *LIPICs*, pages 32:1–32:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPICs.ESA.2020.32.
- 6 Maxime Crochemore, Costas S. Iliopoulos, Marcin Kubica, Jakub Radoszewski, Wojciech Rytter, and Tomasz Waleń. Extracting powers and periods in a word from its runs structure. *Theoretical Computer Science*, 521:29–41, 2014. doi:10.1016/j.tcs.2013.11.018.
- 7 Maxime Crochemore, Costas S. Iliopoulos, Jakub Radoszewski, Wojciech Rytter, Juliusz Straszyński, Tomasz Waleń, and Wiktor Zuba. Internal quasiperiod queries. In Christina Boucher and Sharma V. Thankachan, editors, *String Processing and Information Retrieval - 27th International Symposium, SPIRE 2020, Orlando, FL, USA, October 13-15, 2020, Proceedings*, volume 12303 of *Lecture Notes in Computer Science*, pages 60–75. Springer, 2020. doi:10.1007/978-3-030-59212-7_5.
- 8 Maxime Crochemore, Costas S. Iliopoulos, Jakub Radoszewski, Wojciech Rytter, Juliusz Straszyński, Tomasz Waleń, and Wiktor Zuba. Shortest covers of all cyclic shifts of a string. *Theoretical Computer Science*, 866:70–81, 2021. doi:10.1016/j.tcs.2021.03.011.
- 9 Maxime Crochemore and Wojciech Rytter. Squares, cubes, and time-space efficient string searching. *Algorithmica*, 13(5):405–425, 1995. doi:10.1007/BF01190846.
- 10 Patryk Czajka and Jakub Radoszewski. Experimental evaluation of algorithms for computing quasiperiods. *Theoretical Computer Science*, 854:17–29, 2021. doi:10.1016/j.tcs.2020.11.033.
- 11 Martin Farach. Optimal suffix tree construction with large alphabets. In *38th Annual Symposium on Foundations of Computer Science, FOCS '97, Miami Beach, Florida, USA, October 19-22, 1997*, pages 137–143. IEEE Computer Society, 1997. doi:10.1109/SFCS.1997.646102.
- 12 Nathan J. Fine and Herbert S. Wilf. Uniqueness theorems for periodic functions. *Proceedings of the American Mathematical Society*, 16(1):109–114, 1965. URL: <http://www.jstor.org/stable/2034009>.
- 13 Aviezri S. Fraenkel and Jamie Simpson. How many squares can a string contain? *Journal of Combinatorial Theory, Series A*, 82(1):112–120, 1998. doi:10.1006/jcta.1997.2843.
- 14 Costas S. Iliopoulos, Dennis W. G. Moore, and William F. Smyth. A characterization of the squares in a Fibonacci string. *Theoretical Computer Science*, 172(1-2):281–291, 1997. doi:10.1016/S0304-3975(96)00141-7.
- 15 Tomasz Kociumaka, Marcin Kubica, Jakub Radoszewski, Wojciech Rytter, and Tomasz Waleń. A linear-time algorithm for seeds computation. *ACM Transactions on Algorithms*, 16(2):27:1–27:23, 2020. doi:10.1145/3386369.
- 16 Tomasz Kociumaka, Jakub Radoszewski, Wojciech Rytter, and Tomasz Waleń. Internal pattern matching queries in a text and applications. In Piotr Indyk, editor, *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015*, pages 532–551. SIAM, 2015. doi:10.1137/1.9781611973730.36.

- 17 Roman M. Kolpakov and Gregory Kucherov. Finding maximal repetitions in a word in linear time. In *40th Annual Symposium on Foundations of Computer Science, FOCS '99, 17-18 October, 1999, New York, NY, USA*, pages 596–604. IEEE Computer Society, 1999. doi:10.1109/SFCS.1999.814634.
- 18 Yin Li and William F. Smyth. Computing the cover array in linear time. *Algorithmica*, 32(1):95–106, 2002. doi:10.1007/s00453-001-0062-2.
- 19 Dennis W. G. Moore and William F. Smyth. A correction to “An optimal algorithm to compute all the covers of a string”. *Information Processing Letters*, 54(2):101–103, 1995. doi:10.1016/0020-0190(94)00235-Q.