Efficient Enumeration of Distinct Factors Using Package Representations

Panagiotis Charalampopoulos[1,2,⋆][0000−0002−6024−1557], Tomasz Kociumaka[3,⋆⋆][0000−0002−2477−1702], Jakub Radoszewski[2,⋆⋆⋆][0000−0002−0067−6401], Wojciech Rytter[2][0000−0002−9162−6724], Tomasz Waleń[2,⋆⋆⋆][0000−0002−7369−3309], and Wiktor Zuba[2,⋆⋆⋆][0000−0002−1988−3507]

[1] Department of Informatics, King's College London, UK,
panagiotis.charalampopoulos@kcl.ac.uk
[2] Institute of Informatics, University of Warsaw, Poland
{jrad,rytter,walen,w.zuba}@mimuw.edu.pl
[3] Department of Computer Science, Bar-Ilan University, Ramat Gan, Israel
kociumaka@mimuw.edu.pl

Abstract. We investigate properties and applications of a new compact representation of string factors: families of *packages*. In a string T, each package (i, ℓ, k) represents the factors of T of length ℓ that start in the interval $[i, i + k]$. A family $\mathcal{F}$ of packages represents the set $\mathsf{Factors}(\mathcal{F})$ defined as the union of the sets of factors represented by individual packages in $\mathcal{F}$. We show how to efficiently enumerate $\mathsf{Factors}(\mathcal{F})$ and showcase that this is a generic tool for enumerating important classes of factors of T, such as powers and antipowers. Our approach is conceptually simpler than problem-specific methods and provides a unifying framework for such problems, which we hope can be further exploited.

We also consider a special case of the problem in which every occurrence of every factor represented by $\mathcal{F}$ is captured by some package in $\mathcal{F}$. For both applications mentioned above, we construct an efficient package representation that satisfies this property.

We develop efficient algorithms that, given a family $\mathcal{F}$ of m packages in a string of length n, report all distinct factors represented by these packages in $\mathcal{O}(n \log^2 n + m \log n + |\mathsf{Factors}(\mathcal{F})|)$ time for the general case and in the optimal $\mathcal{O}(n + m + |\mathsf{Factors}(\mathcal{F})|)$ time for the special case. We can also compute $|\mathsf{Factors}(\mathcal{F})|$ in $\mathcal{O}(n \log^2 n + m \log n)$ time in the general case and in $\mathcal{O}(n + m)$ time in the special case.

In particular, we improve over the state-of-the-art $\mathcal{O}(nk^4 \log k \log n)$-time algorithm for computing the number of distinct k-antipower factors, by providing an algorithm that runs in $\mathcal{O}(nk^2)$ time, and we obtain an alternative linear-time algorithm to enumerate distinct squares.

⋆ Partially supported by ERC grant TOTAL under the EU's Horizon 2020 Research and Innovation Programme (agreement no. 677651).

⋆⋆ Supported by ISF grants no. 1278/16 and 1926/19, by a BSF grant no. 2018364, and by an ERC grant MPM under the EU's Horizon 2020 Research and Innovation Programme (grant no. 683064).

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

Keywords: square in a string · antipower · longest previous factor array · string synchronising set

1 Introduction

There are many interesting subsets of factors of a given string T of length n which can be described very concisely (sometimes in $\mathcal{O}(n)$ space, even for subsets of quadratic size). In this paper, we consider compact descriptions, called *package representations*, defined in terms of weighted intervals: each interval $[i, i+k]$ gives starting positions of factors and the weight ℓ gives the common length of these factors. Formally, $\mathcal{F}$ is a set of triples (i, ℓ, k).

By $\mathsf{Factors}(\mathcal{F})$ we denote the set of factors in a given text T of length n that are represented by packages from $\mathcal{F}$. More formally,

$$\mathsf{Factors}(\mathcal{F}) = \{T[j\,..\,j+\ell) : j \in [i, i+k] \text{ and } (i, \ell, k) \in \mathcal{F}\}.$$

A package representation $\mathcal{F}$ is called *special* if it represents all occurrences of $\mathsf{Factors}(\mathcal{F})$. Formally, $\mathcal{F}$ is special if for every factor $F \in \mathsf{Factors}(\mathcal{F})$ and for every occurrence $T[j\,..\,j+\ell) = F$, there is a triple $(i, \ell, k) \in \mathcal{F}$ such that $j \in [i, i+k]$. Special representations describe *all* occurrences of factors with a given property.

We consider the following subsets of factors.

Powers. A square is a string of the form X^2. In general, for an integer $k > 1$, a k-power is a string of the form X^k. This notion can be generalized to rational exponents $\gamma > 1$, setting $X^\gamma = X^k X[1\,..\,r]$ for $\gamma = k + r/|X|$, where k and $r < |X|$ are non-negative integers.

Antipowers. A k-antipower (for an integer $k \geq 2$) is a concatenation of k pairwise distinct strings of the same length. Antipowers were introduced in [15] and have already attracted considerable attention [1,2,4,14,25].

Example 1. Consider a string $T = \texttt{abababababa}$. The squares in T can be represented by a set of packages $\mathcal{F} = \{(1, 4, 7), (1, 8, 3)\}$. The package $(1, 4, 7)$ represents all the squares of length 4 and the package $(1, 8, 3)$—those of length 8.

Our problem can be related to computing the *subword complexity* of the string T; see, e.g., [31]. Let us recall that the subword complexity is a function which gives, for every $\ell \in [1, n]$, the number of different factors of T of length ℓ. The subword complexity of a given string can be computed using the suffix tree in linear time. Our algorithm can be easily augmented to determine, for each length ℓ, the number of length-ℓ factors in $\mathsf{Factors}(\mathcal{F})$.

Our results. We compute $|\mathsf{Factors}(\mathcal{F})|$ in $\mathcal{O}(n \log^2 n + m \log n)$ time in the general case and in $\mathcal{O}(n + m)$ time in the special case, both for any length-n string T over an integer alphabet. The solution to the general case uses string synchronising sets and runs, whereas the solution to the special case is based on the longest previous factor array. Our algorithms for special package representations

yield new simple algorithms for reporting and counting powers and antipowers. In particular, we present the first linear-time algorithms to count and enumerate distinct γ-powers for a given rational constant $\gamma > 1$; Crochemore et al. [11] showed how to do this for integer γ only. For k-antipowers, we improve the previously known best time complexity.

2 Algorithms for Special Package Representations

Let $T = T[1] \cdots T[n]$. The longest previous factor array $LPF[1\,..\,n]$ is defined as

$$LPF[i] = \max\{\ell \geq 0 \,:\, T[i\,..\,i+\ell) = T[j\,..\,j+\ell) \text{ for some } j \in [1, i-1]\}.$$

This array can be computed in $\mathcal{O}(n)$ time [9,10]. Let

$$\begin{aligned} U_\ell \;&=\; \{\, j \in [1,n] \;:\; LPF[j] \geq \ell \,\} \\ \mathsf{Pairs}(\mathcal{F}) \;&=\; \bigcup_{(i,\ell,k)\,\in\mathcal{F}} \{(j,\ell) \;:\; j \in [i, i+k] \setminus U_\ell\}. \end{aligned}$$

The algorithms are based on the following crucial observation that links the solution to the special case with the LPF table.

Observation 2. *If $\mathcal{F}$ is a special package representation, then* $\mathsf{Factors}(\mathcal{F}) = \{\, T[j\,..\,j+\ell) \,:\, (j,\ell) \in \mathsf{Pairs}(\mathcal{F}) \,\}$ *and* $|\mathsf{Factors}(\mathcal{F})| = |\mathsf{Pairs}(\mathcal{F})|$.

2.1 Reporting Distinct Factors

Due to Observation 2, reporting all distinct factors reduces to computing the set $\mathsf{Pairs}(\mathcal{F})$. We can assume that packages representing factors of the same length are disjoint; this can be achieved by merging overlapping packages in a preprocessing step that can be executed in $\mathcal{O}(n)$ time using radix sort.

The definition of $\mathsf{Pairs}(\mathcal{F})$ yields the following (inefficient) algorithm. It constructs the sets U_ℓ for all $\ell = n, \ldots, 1$ and, for each of them, generates all elements of the set $\mathsf{Pairs}(\mathcal{F})$ with the second component equal to ℓ.

Algorithm 1: High-level structure of the algorithm.

```
U := ∅; P := ∅
for ℓ := n down to 1 do
    U := U ∪ {j : LPF[j] = ℓ}                          // U = U_ℓ
    foreach (i, ℓ, k) ∈ F do
        foreach j ∈ [i, i + k] \ U do
            P := P ∪ {(j, ℓ)}                          // Ultimately, P = Pairs(F)
```

Next, we describe an efficient implementation of Algorithm 1 based on the union-find data structure. In our algorithm, the elements of the data structure are $[1, n+1]$ and the sets stored in the data structure always form intervals. The

operation $\texttt{Find}(i)$ returns the rightmost element of the interval containing i, and the operation $\texttt{Union}(i)$ joins the intervals containing elements i and $i-1$.

Algorithm 2: Implementation of Algorithm 1.

```
P := ∅
for i := 0 to n + 1 do Create set {i}
for ℓ := n down to 1 do
    foreach j such that LPF[j] = ℓ do Union(j)
    foreach (i, ℓ, k) ∈ F do
        j := Find(i − 1) + 1
        while j ≤ i + k do
            P := P ∪ {(j, ℓ)}
            j := Find(j) + 1
```

Theorem 3. *In the case of special package representations, all elements of* $\mathsf{Factors}(\mathcal{F})$ *can be reported (without duplicates) in* $\mathcal{O}(n+m+|\mathsf{Factors}(\mathcal{F})|)$ *time.*

Proof. We use Algorithm 2. The set U_ℓ is stored in the union-find data structure so that for each interval $[i,j]$ in the data structure, $i \notin U_\ell$ and $[i+1,j] \subseteq U_\ell$.

The elements of $\mathcal{F}$ are sorted by the second component using radix sort. The union-find data structure admits at most n union operations and $m+|\mathsf{Factors}(\mathcal{F})|$ find operations. We use a data structure for a special case of the union-find problem, where the sets of the partition have to form integer intervals at all times, so that each operation takes $\mathcal{O}(1)$ amortized time [18]. □

2.2 Counting Distinct Factors

Let us start with a warm-up algorithm. Recall that, in a preprocessing, we made sure that packages representing factors of the same length are disjoint.

By Observation 2, for each $(i,\ell,k) \in \mathcal{F}$, it suffices to count the number of elements in $LPF[i\,..\,i+k]$ that are smaller than ℓ. This can be done using range queries in time $O((n+m)\sqrt{\log n})$.

Let us proceed to a linear-time algorithm. We start with a simple fact.

Fact 4. *For every length-n text T, we have* $\sum_{i=1}^{n-1} |LPF[i+1]-LPF[i]| = \mathcal{O}(n)$.

Proof. The claim follows from the fact that $LPF[i+1] \geq LPF[i]-1$ for $i \in [1,n-1]$. To prove this inequality, let $\ell = LPF[i]$. We have $T[i\,..\,i+\ell) = T[j\,..\,j+\ell)$ for some $j < i$. Hence, $T[i+1\,..\,i+\ell) = T[j+1\,..\,j+\ell)$, so $LPF[i+1] \geq \ell-1$. □

We reduce the counting problem to answering off-line a linear number of certain queries. The off-line structure of the computation is crucial for efficiency.

Theorem 5. *In the case of special package representations,* $|\mathsf{Factors}(\mathcal{F})|$ *can be computed in* $\mathcal{O}(n+m)$ *time.*

Proof. Consider the following queries:

$$Q(i,\ell) = |\,[1,i] \setminus U_\ell\,| = |\{j \in [1,i] : LPF[j] < \ell\}|.$$

Then, the counting version of our problem reduces to efficiently answering such queries. Indeed, by Observation 2, we have

$$|\mathsf{Factors}(\mathcal{F})| = \sum_{(i,\ell,k)\in\mathcal{F}} Q(i+k,\ell) - Q(i-1,\ell).$$

Thus, we have to answer $\mathcal{O}(m)$ queries of the form $Q(i,\ell)$. An off-line algorithm answering q queries in $\mathcal{O}(n+q)$ time would be sufficient for our purposes.

We maintain an array $A[1\,..\,n]$ such that during the ith phase of the algorithm:

$$A[\ell] = \begin{cases} i - Q(i,\ell) & \text{if } \ell > LPF[i], \\ Q(i,\ell) & \text{otherwise.} \end{cases}$$

Since $LPF[1] = 0$, the array needs to be filled with 1's for the first phase. Next, we observe that $i+1-Q(i+1,\ell) = i-Q(i,\ell)$ if $\ell > \max(LPF[i+1], LPF[i])$ and $Q(i+1,\ell) = Q(i,\ell)$ if $\ell \le \min(LPF[i+1], LPF[i])$. Hence, in the transition from the ith phase to the $(i+1)$th phase, we only need to update $\mathcal{O}(|LPF[i+1]-LPF[i]|)$ entries of A. By Fact 4, the cost of maintaining the array A for $i = 1$ to n is $\mathcal{O}(n)$ in total. Each query $Q(i,\ell)$ can be answered in $\mathcal{O}(1)$ time during the ith phase.

Consequently, we can answer off-line q queries $Q(i,\ell)$ in $\mathcal{O}(n+q)$ time, assuming that the queries are sorted by the first component. Sorting can be performed in $\mathcal{O}(n+q)$ time using radix sort. □

3 Applications

In this section, we show three applications of special package representations.

3.1 Squares

It is known that a string of length n contains at most $\frac{11}{6}n$ distinct squares [12,17], and the same bound hold for γ-powers with $\gamma \ge 2$. Moreover, all the distinct square factors in a string over an integer alphabet can be reported in $\mathcal{O}(n)$ time [6,11,20]. The algorithm from [11] can report distinct string powers of a given integer exponent using a run-based approach via Lyndon roots. Hence, it can report distinct squares and cubes in particular. We show that our generic approach—which is also much simpler—applies to this problem.

A *generalised run* in a string T is a triple (i,j,p) such that:

- $T[i\,..\,j]$ has a period p (not necessarily the shortest) with $2p \le j-i+1$,
- $T[i-1] \neq T[i-1+p]$ if $i > 1$, and $T[j+1] \neq T[j+1-p]$ if $j < n$.

A *run* is a generalised run for which p is the shortest period of $T[i\,..\,j]$. The number of runs and generalised runs is $\mathcal{O}(n)$ and they can all be computed in $\mathcal{O}(n)$ time; see [5,30].

Proposition 6. *All distinct squares in a string of length n can be computed in $\mathcal{O}(n)$ time.*

Proof. A generalised run (i, j, p) induces squares $T[k \,..\, k + 2p)$ for all $k \in [i, j - 2p + 1]$. Moreover, each occurrence of a square is induced by exactly one generalised run. For every generalised run (i, j, p), we add package $(i, 2p, j - 2p - i + 1)$ to $\mathcal{F}$; see Fig. 1. Then, we solve the factors problem using Theorem 3. □

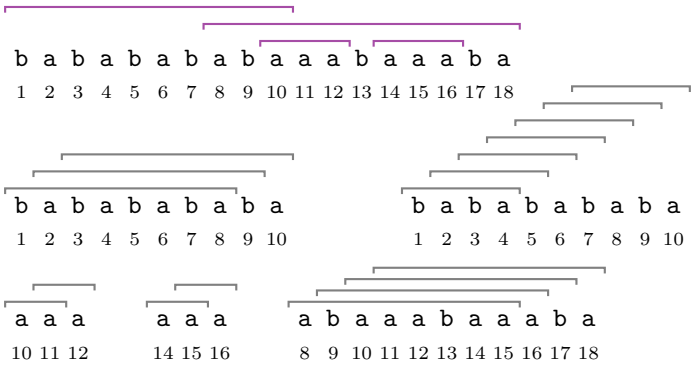

Fig. 1. Four runs (presented at the top) generate a package representation (below) of all squares as a set of five packages: $\{(1, 8, 2), (1, 4, 6), (10, 2, 1), (14, 2, 1), (8, 8, 3)\}$. One of the runs induces two generalised runs: with periods 2 and 4.

3.2 Powers with Rational Exponents

Proposition 6 can be easily generalised to powers of arbitrary exponent $\gamma \geq 2$. For exponents $\gamma < 2$, however, we need α-gapped repeats apart from the generalised runs. An *α-gapped repeat* (for $\alpha \geq 1$) in a string T is a quadruple (i_1, j_1, i_2, j_2) such that $i_1 \leq j_1 < i_2 \leq j_2$, and the factors $T[i_1 \,..\, j_1] = T[i_2 \,..\, j_2] = U$ and $T[j_1 + 1 \,..\, i_2 - 1] = V$ satisfy $|UV| \leq \alpha|U|$. The two occurrences of U are called the *arms* of the α-gapped repeat and $|UV|$ is called the *period* of the α-gapped repeat. In other words, a gapped repeat is a string $S = T[i_1 \,..\, j_2]$ associated with one of its periods larger than $\frac{1}{2}|S|$. Consequently, the same factor $T[i_1 \,..\, j_2]$ can induce many α-gapped repeats.

An α-gapped repeat is called *maximal* if its arms cannot be extended simultaneously with the same character to either direction. The number of maximal α-gapped repeats in a string of length n is $\mathcal{O}(n\alpha)$ and they can all be computed in $\mathcal{O}(n\alpha)$ time assuming an integer alphabet [19].

Theorem 7. *For a given rational number $\gamma > 1$, all distinct γ-powers in a length-n string can be counted in $\mathcal{O}(\frac{\gamma}{\gamma-1}n)$ time and enumerated in $\mathcal{O}(\frac{\gamma}{\gamma-1}n + \mathsf{output})$ time.*

Proof. Each γ-power X^γ with $\gamma < 2$ is a $\frac{1}{\gamma-1}$-gapped repeat with period $|X|$, and therefore it is contained in a maximal $\frac{1}{\gamma-1}$-gapped repeat or in a generalised run with the same period; see [29]. Moreover, each γ-power X^γ with $\gamma \geq 2$ is contained in a generalised run with period $|X|$.

In other words, to generate all γ-powers, for each generalised run and $\frac{1}{\gamma-1}$-gapped repeat (if $\gamma < 2$) with period p, we need to consider all factors contained in it of length γp, provided that γp is an integer; see Fig. 2.

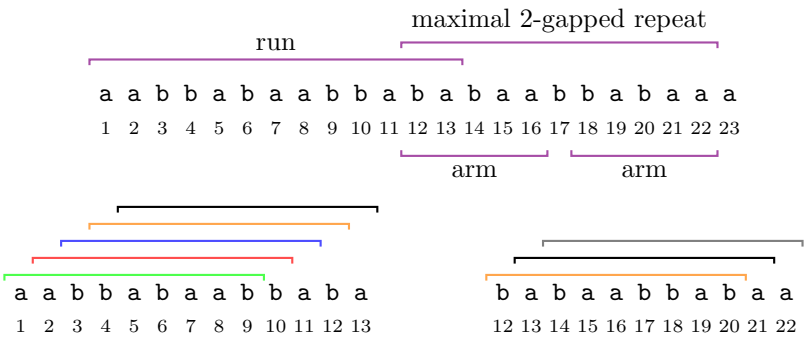

Fig. 2. A string with a (generalised) run with period 6 and a maximal $\frac{6}{5}$-gapped repeat (hence, also a maximal 1.5-gapped repeat) with period 6. The run and the gapped repeat generate 1.5-powers of length 9. Equal 1.5-powers are drawn with the same color; in total, the string contains 6 distinct 1.5-powers of length 9.

We proceed as follows. For each generalised run (i, j, p), if γp is an integer and $j - i + 1 \geq \gamma p$, then we insert $(i, \gamma p, j - i + 1 - \gamma p)$ to $\mathcal{F}$. Moreover, if $\gamma < 2$, then for each maximal $\frac{1}{\gamma-1}$-gapped repeat (i_1, j_1, i_2, j_2) with period $p = i_2 - i_1$, if γp is an integer, then we insert $(i_1, \gamma p, j_2 - i_1 + 1 - \gamma p)$ to $\mathcal{F}$. By the above discussion, the constructed family $\mathcal{F}$ is a special package representation of all γ-powers. The claim follows by Theorems 3 and 5. □

Remark 8. For every fixed rational number $\gamma < 2$, strings of length n may contain $\Omega(n^2)$ distinct γ-powers. Specifically, if $\gamma = 2 - \frac{x}{y}$, where x and y are coprime positive integers, then the number of γ-powers in $\mathtt{a}^m\mathtt{ba}^m$ is $\Theta(\frac{m^2 x}{y^2})$ [28].

3.3 Antipowers

In [25], it was shown how to report all occurrences of k-antipowers in $\mathcal{O}(nk \log k + \mathsf{output})$ time and count them in $\mathcal{O}(nk \log k)$ time. In [26], it was shown that the number of distinct k-antipower factors in a string of length n can be computed in $\mathcal{O}(nk^4 \log k \log n)$ time. Below, we show how to improve the latter result.

Theorem 9. *All distinct k-antipower factors of a string of length n can be reported in $\mathcal{O}(nk^2 + \mathsf{output})$ time and counted in $\mathcal{O}(nk^2)$ time.*

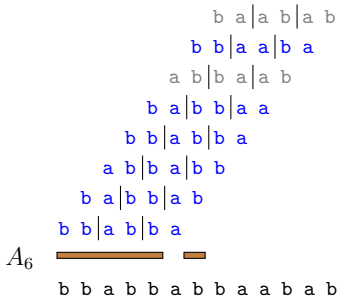

Fig. 3. Here, we consider 3-antipowers of length $\ell = 6$. The set of their starting positions is $A_6 = [1, 5] \cup [7, 7]$. Note that the first and the fourth antipower are the same, so we have only 5 distinct 3-antipowers of length 6. Interestingly $A_\ell = \emptyset$ for $\ell \neq 6$. Hence, the total number of distinct 3-antipowers equals 5.

Proof. The interval representation of a set $A \subseteq [1, n]$ is a collection of all maximal intervals in A. Let A_ℓ be the interval representation of the set of k-antipower factors of T of length ℓ (it can be non-empty only if k divides ℓ); see Fig. 3. In [25, Lemma 13], it was shown that the total size of the interval representations of sets $A_1, \ldots, A_n$ is $\mathcal{O}(nk^2)$. Moreover, they can be computed in $\mathcal{O}(nk^2)$ time.

For each ℓ and each interval $[i, j] \in A_\ell$, we insert $(i, \ell, j - i)$ to $\mathcal{F}$. The conclusion follows by Theorems 3 and 5. □

4 Enumerating General Package Representations

For most of this section, we will focus on computing $|\mathsf{Factors}(\mathcal{F})|$. In the end, we will briefly explain how our solution can be adapted to enumerate $\mathsf{Factors}(\mathcal{F})$.

We consider highly periodic and non-highly-periodic factors separately (a precise definition follows). In both cases, we will employ the solution of Kociumaka et al. [26] for the so-called Path Pairs Problem, which we define below.

We say that $\mathcal{T}$ is a *compact tree* if it is a rooted tree with positive integer weights on edges. If an edge weight is $e > 1$, this edge contains $e - 1$ implicit nodes. A *path* in a compact tree is an upwards or downwards path that connects two explicit nodes.

Path Pairs Problem

Input: Two compact trees $\mathcal{T}$ and $\mathcal{T}'$ containing up to N explicit nodes each, and a set Π of M pairs (π, π') of equal-length paths, where π is a path going downwards in $\mathcal{T}$ and π' is a path going upwards in $\mathcal{T}'$.

Output: $|\bigcup_{(\pi,\pi')\in\Pi} \mathsf{Induced}(\pi, \pi')|$, where by $\mathsf{Induced}(\pi, \pi')$ we denote the set of pairs of (explicit or implicit) nodes (u, u') such that, for some i, the ith node on π is u and the ith node on π' is u'.

Lemma 10 ([26]). *The* Path Pairs Problem *can be solved in time* $\mathcal{O}(N + M \log N)$ *assuming that the weighted heights of the input trees do not exceed* N.

4.1 Non-Highly-Periodic Factors

Our solution uses the string synchronising sets recently introduced by Kempa and Kociumaka [22].

Informally, in the simpler case that T is cube-free, a τ-synchronising set of T consists in a small set of positions of T, called here *synchronisers*, such that each length-τ fragment of T contains at least one synchroniser, and the synchronisers within two long enough matching fragments of T are consistent.

Formally, for a string T and a positive integer $\tau \leq \frac{1}{2}n$, a set $S \subseteq [1, n-2\tau+1]$ is a *τ-synchronising set* of T if it satisfies the following two conditions:

1. If $T[i\,..\,i+2\tau) = T[j\,..\,j+2\tau)$, then $i \in S$ if and only if $j \in S$.
2. For $i \in [1, n-3\tau+2]$, $S \cap [i\,..\,i+\tau) = \emptyset$ if and only if $\mathsf{per}(T[i\,..\,i+3\tau-2]) \leq \frac{1}{3}\tau$.

Theorem 11 ([22]). *Given a string T of length n over an integer alphabet and a positive integer $\tau \leq \frac{1}{2}n$, one can construct in $\mathcal{O}(n)$ time a τ-synchronising set of T of size $\mathcal{O}(\frac{n}{\tau})$.*

As in [22], for a τ-synchronising set S, let $\mathsf{succ}_S(i) := \min\{j \in S \cup \{n-2\tau+2\} : j \geq i\}$ and $\mathsf{pred}_S(i) := \max\{j \in S \cup \{0\} : j \leq i\}$.

Lemma 12 ([22]). *If a factor U of T with $|U| \geq 3\tau - 1$ and $\mathsf{per}(U) > \frac{1}{3}\tau$ occurs at positions i and j in T, then $\mathsf{succ}_S(i) - i = \mathsf{succ}_S(j) - j \leq |U| - 2\tau$.*

By U^R we denote the reversal of a string U. We show the following result.

Lemma 13 (Aperiodic Lemma). *Assume that we are given a text T of length n, a positive integer $x \leq \frac{1}{3}n$, and a family $\mathcal{F}$ of m packages that represent factors of lengths in $[3x, 9x)$ and shortest periods greater than $\frac{1}{3}x$. Then, $|\mathsf{Factors}(\mathcal{F})|$ can be computed in $\mathcal{O}((n+m)\log n)$ time.*

Proof. We compute an x-synchronising set S of T in $\mathcal{O}(n)$ time using Theorem 11 and build the suffix trees $\mathcal{T}$ and $\mathcal{T}'$ of T and T^R, respectively, in $\mathcal{O}(n)$ time [13].

Let us now focus on all packages representing factors of a fixed length ℓ. By relying on Lemma 12, we will intuitively assign each factor to its first synchroniser.

Let us denote $\mathcal{A}_\ell = \bigcup\{[i, i+k] : (i, \ell, k) \in \mathcal{F}\}$. For each $j \in \mathcal{A}_\ell$, let $s = \mathsf{succ}_S(j)$ and consider $P_j = T[j\,..\,s]$ and $Q_j = [s+1\,..\,j+\ell)$; see Fig. 4.

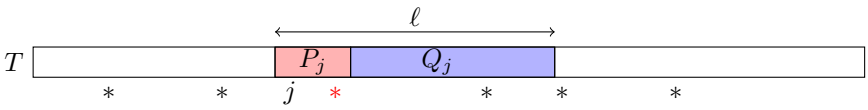

Fig. 4. The elements of an x-synchronising set S of string T are denoted by asterisks. The position j is an element of $[i, i+k]$ for some package (i, ℓ, k). The red asterisk denotes the synchroniser $s = \mathsf{succ}_S(j)$.

Note that, by Lemma 12, $s-j \leq x$ and, as $j \leq n-\ell+1$, we have $s \leq n-2x+1$. Thus, $s \in S$. Hence, Lemma 12 implies that, for any $j, j' \in \mathcal{A}_\ell$ such that $T[j\,..\,j+\ell) = T[j'\,..\,j'+\ell)$, we have $P_j = P_{j'}$ and $Q_j = Q_{j'}$. Consequently, our problem reduces to computing the size of the set $\mathcal{P}_\ell = \{(P_j, Q_j) : j \in \mathcal{A}_\ell\}$. In turn, in our instance of the Path Pairs Problem, we want to count the pairs of nodes $u \in \mathcal{T}'$, $v \in \mathcal{T}$ such that $(\mathcal{L}(u)^R, \mathcal{L}(v)) \in \mathcal{P}_\ell$, where $\mathcal{L}(u)$ is the label of the path from the root to the node u. It remains to show how to compute path pairs that induce exactly these pairs of nodes. To this end, we design a line-sweeping algorithm.

We initialize an empty set Π that will eventually store the desired pairs of paths. We will scan the text T in a left-to-right manner with two fingers: f_p for packages and f_s for synchronisers, both initially set to 0. We maintain an invariant that $f_p \leq f_s$. Whenever $f_p - 1 = f_s \leq n-2x$, we set $f_s = \mathsf{succ}_S(f_s+1)$.

The finger f_p is repeatedly incremented until it reaches f_s. For each maximal interval $[i,j] \subseteq \mathcal{A}_\ell$ that f_p encounters, we do the following: If $j > f_s$, we split the interval into $[i, i+f_s]$ and $[f_s+1, j]$ and consider the first of them as $[i,j]$. Let

$$X_1 = T[i\,..\,f_s],\ X_2 = T[j\,..\,f_s],\ Y_1 = T[f_s+1\,..\,i+\ell),\ Y_2 = T[f_s+1\,..\,j+\ell).$$

For $k = 1, 2$, let u_k be the locus of X_k^R in $\mathcal{T}'$ and v_k be the locus of Y_k in $\mathcal{T}$. If either of these loci is an implicit node, we make it explicit. Finally, we add to Π the pair of paths u_1-to-u_2 in $\mathcal{T}'$ and v_1-to-v_2 in $\mathcal{T}$.

Let us denote the number of packages representing factors of length ℓ by m_ℓ. As there are $\mathcal{O}(\frac{n}{x})$ synchronisers, the line-sweeping algorithm can be performed in $\mathcal{O}(\frac{n}{x} + m_\ell)$ time. Thus, the number of paths (and extra explicit nodes) that we introduce in the two suffix trees is also $\mathcal{O}(\frac{n}{x} + m_\ell)$.

Over all $\ell \in [3x, 9x)$, we have

$$\mathcal{O}\left(x \cdot \frac{n}{x} + \sum_{\ell=3x}^{9x-1} m_\ell\right) = \mathcal{O}(n+m)$$

pairs of paths. The only operations that we need to explain how to perform efficiently are (a) computing the loci of strings in $\mathcal{T}$ and $\mathcal{T}'$ and (b) making all of them explicit. Part (a) can be implemented using an efficient algorithm for answering a batch of weighted ancestor queries from [24]. In part (b), we process the weighted ancestors in an order of non-decreasing weights, after globally sorting them using radix sort. The whole construction works in $\mathcal{O}(n+m)$ time. We obtain an instance of the Path Pairs Problem with $N, M = \mathcal{O}(n+m)$. The suffix trees are of weighted height $\mathcal{O}(n)$, so Lemma 10 completes the proof. □

4.2 Highly Periodic Factors

A string U is called *periodic* if $2 \cdot \mathsf{per}(U) \leq |U|$ and *highly periodic* if $3 \cdot \mathsf{per}(U) \leq |U|$.

The *Lyndon root* of a periodic string U is the lexicographically smallest rotation of its length-$\mathsf{per}(U)$ prefix. If L is the Lyndon root of a periodic string

U, then U can be uniquely represented as (L, y, a, b) for $0 \le a, b < |L|$ such that $U = L[|L| - a + 1\,..\,|L|]L^y L[1\,..\,b]$. We call this the *Lyndon representation* of U.

In $\mathcal{O}(n)$ time, one can compute the Lyndon representations of all runs [11]. The unique run that extends a periodic factor of T can be computed in $\mathcal{O}(1)$ time after $\mathcal{O}(n)$-time preprocessing [5,27]. This allows computing its Lyndon representation in $\mathcal{O}(1)$ time.

For highly periodic factors, we will use *Lyndon roots* instead of synchronisers. The rest of this subsection is devoted to proving the following lemma.

Lemma 14 (Periodic Lemma). *Given a text T of length n and a set $\mathcal{F}$ of m packages of highly periodic factors, $|\mathsf{Factors}(\mathcal{F})|$ can be computed in $\mathcal{O}((n + m)\log n)$ time.*

Proof. For each $(i, \ell, k) \in \mathcal{F}$ and $j \in [i, i + k]$, the fragment $T[j\,..\,j + \ell)$ has a (unique) Lyndon representation (L, y, a, b) for some Lyndon root L. Let

$$P_{j,\ell} = L[|L| - a + 1\,..\,|L|] \text{ and } Q_{j,\ell} = L^y L[1\,..\,b]$$

Our problem consists in computing the size of the set

$$\mathcal{P} = \{(P_{j,\ell}, Q_{j,\ell}) : j \in [i, i + k],\ (i, \ell, k) \in \mathcal{F}\}$$

Let $\mathcal{T}$ and $\mathcal{T}'$ be the suffix trees of T and T^R, respectively. Then, we want to compute the number of pairs of nodes $u \in \mathcal{T}'$ and $v \in T$ with $(\mathcal{L}(u)^R, \mathcal{L}(v)) \in \mathcal{P}$ We will show how this reduces to an instance of the Path Pairs Problem.

We have to appropriately define pairs of paths over $\mathcal{T}$ and $\mathcal{T}'$. Let us note that all the factors that each package of $\mathcal{F}$ represents have the same Lyndon root, since two strings with different periods at most $\frac{1}{3}\ell$ cannot overlap on $\ell - 1$ positions by the Fine and Wilf's periodicity lemma [16].

We initialize an empty set Π that will store pairs of paths. Let us consider a package $(i, \ell, k) \in \mathcal{F}$ such that $T[i\,..\,i + k + \ell)$ is represented by (L, y, a, b). By periodicity, we may focus on the factors starting in the first (at most) $|L|$ positions of $T[i\,..\,i + k + \ell)$.

To this end, let $t = \min\{|L|, k + 1\}$. We will insert at most two paths to Π, specified below:

- Let X_1 be the suffix of L of length a, X_2 be the suffix of L of length $a' = \max\{a - t, 0\}$, $Y_1 = L^\infty[1\,..\,\ell - a]$ and $Y_2 = L^\infty[1\,..\,\ell - a']$.
- If $t > a$, let X'_1 be the suffix of L of length $|L| - 1$ and X'_2 be the suffix of L of length $d = |L| + a - t$, $Y'_1 = L^\infty[1\,..\,\ell - |L| + 1]$ and $Y'_2 = L^\infty[1\,..\,\ell - d]$.

See Fig. 5 for an illustration. It can be readily verified that these pairs of paths induce exactly the required pairs of nodes.

For $k = 1, 2$, let u_k be the locus of X_k^R in $\mathcal{T}'$ and v_k be the locus of Y_k in $\mathcal{T}$. If either of these loci is an implicit node, we make it explicit. Finally, we add to Π the pair of paths u_1-to-u_2 in $\mathcal{T}'$ and v_1-to-v_2 in $\mathcal{T}$. Similarly for X'_ks and Y'_ks.

Let us consider the time complexity of the algorithm. The suffix trees $\mathcal{T}$ and $\mathcal{T}'$ can be computed in $\mathcal{O}(n)$ time [13]. Computing the loci of strings and making

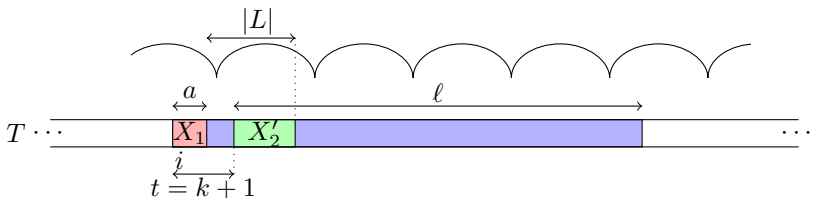

Fig. 5. The shaded part of the text denotes $T[i\,..\,i+k+\ell)$, which can be represented as (L, y, a, b), for some package $(i, \ell, k) \in \mathcal{F}_\ell$. X_1 is shaded in red, while X_2 is the empty string. We are in the case that $t = k + 1 > a$; X_2' is shaded in green.

them explicit can be performed in $\mathcal{O}(n + m)$ time as in the proof of Lemma 13. We obtain an instance of the Path Pairs Problem with $N = \mathcal{O}(n + m)$ and $M \leq 2m$. Lemma 10 completes the proof. □

4.3 Wrap-up

Let $\mathcal{F}_\ell$ be the set of triples from $\mathcal{F}$ with the second component ℓ. Note that one can easily compute the contributions of all packages representing factors whose length is bounded by 2 in $\mathcal{O}(n)$ time using radix sort; we can thus assume that all packages represent factors of length at least 3.

We will iterate over $x = 3^j$ for all integers $j \in [1, \lfloor \log_3 n \rfloor - 1]$. For each $\ell \in [3x, 9x)$, we want to replace $\mathcal{F}_\ell$ by two—not too large—sets of packages:

- $\mathcal{F}_\ell^p$ representing factors with shortest period at most $x/3$, and
- $\mathcal{F}_\ell^a$ representing factors with shortest period greater than $x/3$,

such that $\mathsf{Factors}(\mathcal{F}_\ell) = \mathsf{Factors}(\mathcal{F}_\ell^p) \cup \mathsf{Factors}(\mathcal{F}_\ell^a)$.

Our aim is to decompose each package in $\mathcal{F}_\ell$ in pieces (i.e., decompose $[i, i+k]$ into subintervals), such that all factors represented by each piece either have shortest period at most $x/3$ or none of them does. We then want to group the resulting pieces into the two sets.

Let $\mathcal{R}_x$ denote the set of runs of T with length at least $3x$ and period at most $x/3$. As shown in [23, Section 4.4], $|\mathcal{R}_x| = \mathcal{O}(n/x)$. Further, $\mathcal{R}_x$ can be computed in $\mathcal{O}(n)$ time, by filtering out the runs that do not satisfy the criteria.

Lemma 15. *Given $\mathcal{R}_x$, we can compute sets $\mathcal{F}_\ell^p$ and $\mathcal{F}_\ell^a$ in $\mathcal{O}(n/x + |\mathcal{F}_\ell|)$ time.*

Proof. Initially, let $I = \emptyset$. For each run $R = T[a\,..\,b]$ with $\mathsf{per}(R) \leq x/3$ and $|R| \geq \ell$, we set $I := I \cup [a, b - \ell + 1]$. There are $\mathcal{O}(n/x)$ such runs and hence our representation of I consists of $\mathcal{O}(n/x)$ intervals.

Recall that packages are pairwise disjoint. We decompose a package $(i, \ell, k) \in \mathcal{F}_\ell$ as follows.

For each maximal interval $[r, t]$ in $[i, i+k] \cap I$ we insert $(r, \ell, t-r)$ to $\mathcal{F}_\ell^p$, while for each maximal interval $[r', t']$ in $[i, i+k] \setminus I$ we insert $(r', \ell, t'-r')$ to $\mathcal{F}_\ell^a$. This can be done in $\mathcal{O}(n/x + |\mathcal{F}_\ell|)$ time with a standard line-sweeping algorithm. □

Now, let us put everything together. First of all, we compute $\mathcal{R}_x$ for each $x \in [1, \lfloor \log_3 n \rfloor - 1]$ in $\mathcal{O}(n \log n)$ total time. Then, for each ℓ, we replace $\mathcal{F}_\ell$ by $\mathcal{F}_\ell^p$ and $\mathcal{F}_\ell^a$ in $\mathcal{O}(n/\ell + |\mathcal{F}_\ell|)$ time, employing Lemma 15.

We process all $\mathcal{F}_\ell^p$'s together, as each factor U represented by them must be highly periodic; since $\mathsf{per}(U) \leq x/3$ and $|U| \geq 3x$ for some x, we surely have $3 \cdot \mathsf{per}(U) \leq |U|$. The total size of these sets is $\sum_{\ell=3}^{n} \mathcal{O}(n/\ell + |\mathcal{F}_\ell|) = \mathcal{O}(n \log n + m)$, and hence a call to Lemma 14 requires $\mathcal{O}(n \log^2 n + m \log n)$ time.

Then, we make a call to Lemma 13 for each $x \in [1, \lfloor \log_3 n \rfloor - 1]$, and the union of sets $\mathcal{F}_\ell^a$ for $\ell \in [3x, 9x)$. Again by Lemma 15, for each such ℓ, we have $|\mathcal{F}_\ell^a| = \mathcal{O}(n/x + |\mathcal{F}_\ell|)$.

The total time complexity required by the calls to Lemma 13 is:

$$\begin{aligned}\sum_{x=1}^{\lfloor \log_3 n \rfloor - 1} \mathcal{O}\left(n \log n + \sum_{\ell=3x}^{9x-1} |\mathcal{F}_\ell| \log n\right) &= \mathcal{O}(n \log^2 n) + \sum_{\ell=3}^{n} \mathcal{O}(|\mathcal{F}_\ell| \log n) \\ &= \mathcal{O}(n \log^2 n + m \log n).\end{aligned}$$

We have thus proved the main result of this section.

Theorem 16. *$|\mathsf{Factors}(\mathcal{F})|$ can be computed in $\mathcal{O}(n \log^2 n + m \log n)$ time.*

4.4 Reporting Factors

The reporting version of the Path Pairs Problem, where one is to output $\bigcup_{(\pi,\pi') \in \Pi} \mathsf{Induced}(\pi, \pi')$, can be solved in $\mathcal{O}(N + M \log N + \mathsf{output})$ time by a straightforward modification of the proof of Lemma 10.[1] We can also retrieve a pair of paths inducing each pair of nodes within the same time complexity (in order to be able to represent the relevant string as a factor of T).

Theorem 17. *All elements of $\mathsf{Factors}(\mathcal{F})$ can be reported (without duplicates) in $\mathcal{O}(n \log^2 n + m \log n + \mathsf{output})$ time.*

5 Final Remarks

Another natural representation of factors consists in a set of intervals $\mathcal{I}$, such that each $[i, j] \in \mathcal{I}$ represents all factors of $T[i \,..\, j]$. This problem is very closely related to the problem of property indexing [3,7,8,21]. Employing either of the optimal property indexes that were presented in [7,8], one can retrieve the (number of) represented factors in optimal time.

[1] The workhorse of Lemma 10 is computing the size of the union of certain 1D-intervals. For the reporting version, we simply have to report all elements of this union.

References

1. Alamro, H., Badkobeh, G., Belazzougui, D., Iliopoulos, C.S., Puglisi, S.J.: Computing the antiperiod(s) of a string. In: 30th Annual Symposium on Combinatorial Pattern Matching, CPM 2019. LIPIcs, vol. 128, pp. 32:1–32:11. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2019). https://doi.org/10.4230/LIPIcs.CPM.2019.32
2. Alzamel, M., Conte, A., Greco, D., Guerrini, V., Iliopoulos, C.S., Pisanti, N., Prezza, N., Punzi, G., Rosone, G.: Online algorithms on antipowers and antiperiods. In: String Processing and Information Retrieval - 26th International Symposium, SPIRE 2019. Lecture Notes in Computer Science, vol. 11811, pp. 175–188. Springer (2019). https://doi.org/10.1007/978-3-030-32686-9_13
3. Amir, A., Chencinski, E., Iliopoulos, C.S., Kopelowitz, T., Zhang, H.: Property matching and weighted matching. Theor. Comput. Sci. **395**(2-3), 298–310 (2008). https://doi.org/10.1016/j.tcs.2008.01.006
4. Badkobeh, G., Fici, G., Puglisi, S.J.: Algorithms for anti-powers in strings. Inf. Process. Lett. **137**, 57–60 (2018). https://doi.org/10.1016/j.ipl.2018.05.003
5. Bannai, H., I, T., Inenaga, S., Nakashima, Y., Takeda, M., Tsuruta, K.: The "runs" theorem. SIAM J. Comput. **46**(5), 1501–1514 (2017). https://doi.org/10.1137/15M1011032
6. Bannai, H., Inenaga, S., Köppl, D.: Computing all distinct squares in linear time for integer alphabets. In: 28th Annual Symposium on Combinatorial Pattern Matching, CPM 2017. LIPIcs, vol. 78, pp. 22:1–22:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2017). https://doi.org/10.4230/LIPIcs.CPM.2017.22
7. Barton, C., Kociumaka, T., Liu, C., Pissis, S.P., Radoszewski, J.: Indexing weighted sequences: Neat and efficient. Inf. Comput. **270** (2020). https://doi.org/10.1016/j.ic.2019.104462
8. Charalampopoulos, P., Iliopoulos, C.S., Liu, C., Pissis, S.P.: Property suffix array with applications in indexing weighted sequences. ACM Journal of Experimental Algorithmics **25**(1) (2020). https://doi.org/10.1145/3385898
9. Crochemore, M., Hancart, C., Lecroq, T.: Algorithms on strings. Cambridge University Press (2007)
10. Crochemore, M., Ilie, L.: Computing longest previous factor in linear time and applications. Inf. Process. Lett. **106**(2), 75–80 (2008). https://doi.org/10.1016/j.ipl.2007.10.006
11. Crochemore, M., Iliopoulos, C.S., Kubica, M., Radoszewski, J., Rytter, W., Waleń, T.: Extracting powers and periods in a word from its runs structure. Theor. Comput. Sci. **521**, 29–41 (2014). https://doi.org/10.1016/j.tcs.2013.11.018
12. Deza, A., Franek, F., Thierry, A.: How many double squares can a string contain? Discret. Appl. Math. **180**, 52–69 (2015). https://doi.org/10.1016/j.dam.2014.08.016
13. Farach, M.: Optimal suffix tree construction with large alphabets. In: 38th Annual Symposium on Foundations of Computer Science, FOCS 1997. pp. 137–143. IEEE Computer Society (1997). https://doi.org/10.1109/SFCS.1997.646102
14. Fici, G., Postic, M., Silva, M.: Abelian antipowers in infinite words. Adv. Appl. Math. **108**, 67–78 (2019). https://doi.org/10.1016/j.aam.2019.04.001
15. Fici, G., Restivo, A., Silva, M., Zamboni, L.Q.: Anti-powers in infinite words. J. Comb. Theory, Ser. A **157**, 109–119 (2018). https://doi.org/10.1016/j.jcta.2018.02.009
16. Fine, N.J., Wilf, H.S.: Uniqueness theorems for periodic functions. Proceedings of the American Mathematical Society **16**(1), 109–114 (1965). https://doi.org/10.2307/2034009

17. Fraenkel, A.S., Simpson, J.: How many squares can a string contain? J. Comb. Theory, Ser. A **82**(1), 112–120 (1998). https://doi.org/10.1006/jcta.1997.2843
18. Gabow, H.N., Tarjan, R.E.: A linear-time algorithm for a special case of disjoint set union. J. Comput. Syst. Sci. **30**(2), 209–221 (1985). https://doi.org/10.1016/0022-0000(85)90014-5
19. Gawrychowski, P., I, T., Inenaga, S., Köppl, D., Manea, F.: Tighter bounds and optimal algorithms for all maximal α-gapped repeats and palindromes - finding all maximal α-gapped repeats and palindromes in optimal worst case time on integer alphabets. Theory Comput. Syst. **62**(1), 162–191 (2018). https://doi.org/10.1007/s00224-017-9794-5
20. Gusfield, D., Stoye, J.: Linear time algorithms for finding and representing all the tandem repeats in a string. J. Comput. Syst. Sci. **69**(4), 525–546 (2004). https://doi.org/10.1016/j.jcss.2004.03.004
21. Hon, W., Patil, M., Shah, R., Thankachan, S.V.: Compressed property suffix trees. Inf. Comput. **232**, 10–18 (2013). https://doi.org/10.1016/j.ic.2013.09.001
22. Kempa, D., Kociumaka, T.: String synchronizing sets: sublinear-time BWT construction and optimal LCE data structure. In: 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019. pp. 756–767. ACM (2019). https://doi.org/10.1145/3313276.3316368
23. Kociumaka, T.: Efficient Data Structures for Internal Queries in Texts. Ph.D. thesis, University of Warsaw (2018), `https://mimuw.edu.pl/~kociumaka/files/phd.pdf`
24. Kociumaka, T., Kubica, M., Radoszewski, J., Rytter, W., Waleń, T.: A linear-time algorithm for seeds computation. ACM Trans. Algorithms **16**(2) (2020). https://doi.org/10.1145/3386369
25. Kociumaka, T., Radoszewski, J., Rytter, W., Straszyński, J., Waleń, T., Zuba, W.: Efficient representation and counting of antipower factors in words. In: 13th International Conference on Language and Automata Theory and Applications, LATA 2019. Lecture Notes in Computer Science, vol. 11417, pp. 421–433. Springer (2019). https://doi.org/10.1007/978-3-030-13435-8_31
26. Kociumaka, T., Radoszewski, J., Rytter, W., Straszyński, J., Waleń, T., Zuba, W.: Efficient representation and counting of antipower factors in words (2020), `https://arxiv.org/abs/1812.08101v3`
27. Kociumaka, T., Radoszewski, J., Rytter, W., Waleń, T.: Internal pattern matching queries in a text and applications. In: 26th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015. pp. 532–551. SIAM (2015). https://doi.org/10.1137/1.9781611973730.36
28. Kociumaka, T., Radoszewski, J., Rytter, W., Waleń, T.: String powers in trees. Algorithmica **79**(3), 814–834 (2017). https://doi.org/10.1007/s00453-016-0271-3
29. Kolpakov, R.: Some results on the number of periodic factors in words. Inf. Comput. **270** (2020). https://doi.org/10.1016/j.ic.2019.104459
30. Kolpakov, R.M., Kucherov, G.: Finding maximal repetitions in a word in linear time. In: 40th Annual Symposium on Foundations of Computer Science, FOCS 1999. pp. 596–604. IEEE Computer Society (1999). https://doi.org/10.1109/SFFCS.1999.814634
31. Lothaire, M.: Algebraic Combinatorics on Words. Cambridge University Press (2002). https://doi.org/10.1017/cbo9781107326019