String Periods in the Order-Preserving Model

Garance Gourdel
ENS Paris-Saclay, Cachan, France
garance.gourdel@ens-paris-saclay.fr

Tomasz Kociumaka
University of Warsaw, Warsaw, Poland
kociumaka@mimuw.edu.pl

Jakub Radoszewski
University of Warsaw, Warsaw, Poland
jrad@mimuw.edu.pl

Wojciech Rytter
University of Warsaw, Warsaw, Poland
rytter@mimuw.edu.pl

Arseny Shur
Ural Federal University, Ekaterinburg, Russia
arseny.shur@urfu.ru

Tomasz Waleń
University of Warsaw, Warsaw, Poland
walen@mimuw.edu.pl

Abstract

The order-preserving model (op-model, in short) was introduced quite recently but has already attracted significant attention because of its applications in data analysis. We introduce several types of periods in this setting (op-periods). Then we give algorithms to compute these periods in time $O(n)$, $O(n \log\log n)$, $O(n \log^2 \log n/ \log\log\log n)$, $O(n \log n)$ depending on the type of periodicity. In the most general variant the number of different periods can be as big as $\Omega(n^2)$, and a compact representation is needed. Our algorithms require novel combinatorial insight into the properties of such periods.

2012 ACM Subject Classification Theory of computation → Pattern matching

Keywords and phrases Order-preserving Pattern Matching, Period, Efficient Algorithm

Digital Object Identifier 10.4230/LIPIcs.STACS.2018.38

Related Version A full version of the paper is available at `https://arxiv.org/abs/1801.01404`, [29].

Funding A part of this work was done during Garance Gourdel's internship at University of Warsaw, Poland. Tomasz Kociumaka, Jakub Radoszewski, Wojciech Rytter and Tomasz Waleń were supported by the Polish National Science Center, grant no. 2014/13/B/ST6/00770.

Acknowledgements A part of this work was done during the workshop "StringMasters in Warsaw 2017" that was sponsored by the Warsaw Center of Mathematics and Computer Science. The authors thank the participants of the workshop, especially Hideo Bannai and Shunsuke Inenaga, for helpful discussions.

35th Symposium on Theoretical Aspects of Computer Science (STACS 2018).
Editors: Rolf Niedermeier and Brigitte Vallée; Article No. 38; pp. 38:1–38:16
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany
SYMPOSIUM ON THEORETICAL ASPECTS OF COMPUTER SCIENCE

1 Introduction

Study of strings in the *order-preserving* model (*op-model*, in short) is a part of the so-called non-standard stringology. It is focused on pattern matching and repetition discovery problems in the shapes of number sequences. Here the shape of a sequence is given by the relative order of its elements. The applications of the op-model include finding trends in time series which appear naturally when considering e.g. the stock market or melody matching of two musical scores; see [32]. In such problems periodicity plays a crucial role.

One of motivations is given by the following scenario. Consider a sequence D of numbers that models a time series which is known to repeat the same shape every fixed period of time. For example, this could be certain stock market data or statistics data from a social network that is strongly dependent on the day of the week, i.e., repeats the same shape every consecutive week. Our goal is, given a fragment S of the sequence D, to discover such repeating shapes, called here *op-periods*, in S. We also consider some special cases of this setting. If the beginning of the sequence S is synchronized with the beginning of the repeating shape in D, we refer to the repeating shape as to an *initial* op-period. If the synchronization takes place also at the end of the sequence, we call the shape a *full* op-period. Finally, we also consider *sliding* op-periods that describe the case when every factor of the sequence D repeats the same shape every fixed period of time.

Order-preserving model. Let $[\![a..b]\!]$ denote the set $\{a,\dots,b\}$. We say that two strings $X = X[1]\dots X[n]$ and $Y = Y[1]\dots Y[n]$ over an integer alphabet are *order-equivalent* (*equivalent* in short), written $X \approx Y$, iff $\forall_{i,j\in[\![1..n]\!]}\quad X[i] < X[j] \Leftrightarrow Y[i] < Y[j]$.

▶ **Example 1.** $5\,2\,7\,5\,1\,3\,10\,3\,5 \approx 6\,4\,7\,6\,3\,5\,9\,5\,6$.

Order-equivalence is a special case of a substring consistent equivalence relation (SCER) that was defined in [37].

For a string S of length n, we can create a new string X of length n such that $X[i]$ is equal to the number of distinct symbols in S that are not greater than $S[i]$. The string X is called the *shape* of S and is denoted by $\mathit{shape}(S)$. It is easy to observe that two strings S, T are order-equivalent if and only if they have the same shape.

▶ **Example 2.** $\mathit{shape}(5\,2\,7\,5\,1\,3\,10\,3\,5) = \mathit{shape}(6\,4\,7\,6\,3\,5\,9\,5\,6) = 4\,2\,5\,4\,1\,3\,6\,3\,4$.

Periods in the op-model. We consider several notions of periodicity in the op-model, illustrated by Fig. 1. We say that a string S has a (general) *op-period* p with *shift* $s \in [\![0..p-1]\!]$ if and only if $p < |S|$ and S is a factor of a string $V_1V_2\cdots V_k$ such that:

$$|V_1| = \cdots = |V_k| = p, \quad V_1 \approx \cdots \approx V_k, \quad \text{and } S[s+1..|S|] \text{ is a prefix of } V_2\cdots V_k.$$

The *shape* of the op-period is $\mathit{shape}(V_1)$. One op-period p can have several shifts; to avoid ambiguity, we sometimes denote the op-period as (p, s). We define Shifts_p as the set of all shifts of the op-period p.

An op-period p is called *initial* if $0 \in \mathit{Shifts}_p$, *full* if it is initial and p divides $|S|$, and *sliding* if $\mathit{Shifts}_p = [\![0..p-1]\!]$. Initial and sliding op-periods are particular cases of block-based and sliding-window-based periods for SCER, both of which were introduced in [37].

Models of periodicity. In the standard model, a string S of length n has a period p iff $S[i] = S[i+p]$ for all $i = 1,\dots,n-p$. The famous periodicity lemma of Fine and Wilf [26] states that a "long enough" string with periods p and q has also the period $\gcd(p,q)$. The

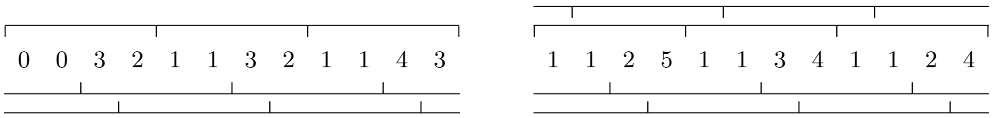

Figure 1 The string to the left has op-period 4 with three shifts: $Shifts_4 = [\![0..0]\!] \cup [\![2..3]\!]$. Due to the shift 0, the string has an initial—therefore, a full—op-period 4. The string to the right has op-period 4 with all four shifts: $Shifts_4 = [\![0..3]\!]$. In particular, 4 is a sliding op-period of the string. Notice that both strings (of length $n = 12$) have (general, sliding) periods 4, but none of them has the order-border (in the sense of [36]) of length $n - 4$.

exact bound of being "long enough" is $p + q - \gcd(p, q)$. This result was generalized to arbitrary number of periods [9, 31, 40].

Periods were also considered in a number of non-standard models. Partial words, which are strings with don't care symbols, possess quite interesting Fine–Wilf type properties, including probabilistic ones; see [4, 5, 6, 38, 39, 30]. In Section 2, we make use of periodicity graphs introduced in [38, 39]. In the abelian (jumbled) model, a version of the periodicity lemma was shown in [15] and extended in [7]. Also, algorithms for computing three types of periods analogous to full, initial, and general op-periods were designed [19, 24, 25, 33, 34, 35]. In the computation of full and initial op-periods we use some number-theoretic tools initially developed in [33, 34]. Remarkably, the fastest known algorithm for computing general periods in the abelian model has essentially quadratic time complexity [19, 35], whereas for the general op-periods we design a much more efficient solution. A version of the periodicity lemma for the parameterized model was proposed in [2].

Op-periods were first considered in [37] where initial and sliding op-periods were introduced and direct generalizations of the Fine–Wilf property to these kinds of op-periods were developed. A few distinctions between the op-periods and periods in other models should be mentioned. First, "to have a period 1" becomes a trivial property in the op-model. Second, all standard periods of a string have the "sliding" property; the first string in Fig. 1 demonstrates that this is not true for op-periods. The last distinction concerns borders. A standard period p in a string S of length n corresponds to a *border* of S of length $n - p$, which is both a prefix and a suffix of S. In the order-preserving setting, an analogue of a border is an *op-border*, that is, a prefix that is equivalent to the suffix of the same length. Op-borders have properties similar to standard borders and can be computed in $O(n)$ time [36]. However, it is no longer the case that a (general, initial, full, or sliding) op-period must correspond to an op-border; see [37].

Previous algorithmic study of the op-model. The notion of order-equivalence was introduced in [32, 36]. (However, note the related combinatorial studies, originated in [22], on containment/avoidance of shapes in permutations.) Both [32, 36] studied pattern matching in the op-model (op-pattern matching) that consists in identifying all consecutive factors of a text that are order-equivalent to a given pattern. We assume that the alphabet is integer and, as usual, that it is polynomially bounded with respect to the length of the string, which means that a string can be sorted in linear time (cf. [16]). Under this assumption, for a text of length n and a pattern of length m, [32] solve the op-pattern matching problem in $O(n + m \log m)$ time and [36] solve it in $O(n + m)$ time. Other op-pattern matching algorithms were presented in [3, 14].

An index for op-pattern matching based on the suffix tree was developed in [18]. For a text of length n it uses $O(n)$ space and answers op-pattern matching queries for a pattern of length m in optimal, $O(m)$ time (or $O(m + Occ)$ time if we are to report all Occ occurrences). The index can be constructed in $O(n \log \log n)$ expected time or $O(n \log^2 \log n / \log \log \log n)$ worst-case time. We use the index itself and some of its applications from [18].

Other developments in this area include a multiple-pattern matching algorithm for the op-model [32], an approximate version of op-pattern matching [28], compressed index constructions [12, 21], a small-space index for op-pattern matching that supports only short queries [27], and a number of practical approaches [8, 10, 11, 13, 23].

Our results. We give algorithms to compute:

- all full op-periods in $O(n)$ time;
- the smallest non-trivial initial op-period in $O(n)$ time;
- all initial op-periods in $O(n \log \log n)$ time;
- all sliding op-periods in $O(n \log \log n)$ expected time or $O(n \log^2 \log n / \log \log \log n)$ worst-case time (and linear space);
- all general op-periods with all their shifts (compactly represented) in $O(n \log n)$ time and space. The output is the family of sets Shifts_p represented as unions of disjoint intervals. The total number of intervals, over all p, is $O(n \log n)$.

In the combinatorial part, we characterize the Fine–Wilf periodicity property (aka interaction property) in the op-model in the case of coprime periods. This result is at the core of the linear-time algorithm for the smallest initial op-period.

Structure of the paper. Combinatorial foundations of our study are given in Section 2. Then in Section 3 we recall known algorithms and data structures for the op-model and develop further algorithmic tools. The remaining sections are devoted to computation of the respective types of op-periods: full and initial op-periods in Section 4, the smallest non-trivial initial op-period in Section 5, all (general) op-periods in Section 6, and sliding op-periods in Section 7. Some proofs have been omitted due to space constraints; they can be found in the preprint [29].

2 Fine–Wilf Property for Op-Periods

The following result was shown as Theorem 2 in [37]. Note that if p and q are coprime, then the conclusion is void, as every string has the op-period 1.

▶ **Theorem 3** ([37]). *Let $p > q > 1$ and $d = \gcd(p, q)$. If a string S of length $n \geq p + q - d$ has initial op-periods p and q, it has initial op-period d. Moreover, if S has length $n \geq p+q-1$ and sliding op-periods p and q, it has sliding op-period d.*

The aim of this section is to show a periodicity lemma in the case that $\gcd(p, q) = 1$.

2.1 Preliminary Notation

For a string S of length n, by $S[i]$ (for $1 \leq i \leq n$) we denote the ith letter of S and by $S[i..j]$ we denote a *factor* of S equal to $S[i] \dots S[j]$. If $i > j$, $S[i..j]$ denotes the empty string ε.

A string which is strictly increasing, strictly decreasing, or constant, is called *strictly monotone*. A *strictly monotone op-period* of S is an op-period with a strictly monotone shape. Such an op-period is called increasing (decreasing, constant) if so is its shape. Clearly, any divisor of a strictly monotone op-period is a strictly monotone op-period as well. A string S is *2-monotone* if $S = S_1 S_2$, where S_1, S_2 are strictly monotone in the same direction.

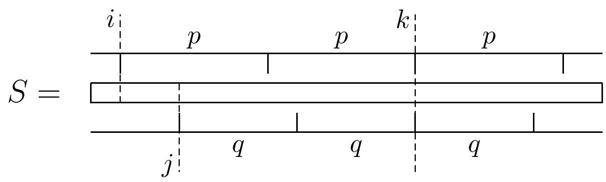

Figure 2 Op-periods (p, i) and (q, j) synchronized at position k.

Below we assume that $n > p > q > 1$. Let a string $S = S[1..n]$ have op-periods (p, i) and (q, j). If there exists a number $k \in [\![1..n-1]\!]$ such that $k \bmod p = i$ and $k \bmod q = j$, we say that these op-periods are *synchronized* and k is a *synchronization point* (see Fig. 2).

▶ Remark. The proof of Theorem 3 can be easily adapted to prove the following.

▶ **Theorem 4.** *Let $p > q > 1$ and $d = \gcd(p, q)$. If op-periods p and q of a string S of length $n \ge p + q - 1$ are synchronized, then S has op-period d, synchronized with them.*

2.2 Periodicity Theorem For Coprime Periods

For a string S, by *trace*(S) we denote a string X of length $|S| - 1$ over the alphabet $\{\texttt{+}, \texttt{0}, \texttt{-}\}$ such that:

$$X[i] = \begin{cases} \texttt{+} & \text{if } S[i] < S[i+1] \\ \texttt{0} & \text{if } S[i] = S[i+1] \\ \texttt{-} & \text{if } S[i] > S[i+1]. \end{cases}$$

▶ **Observation 5.**

(1) *A string is strictly monotone iff its trace is a unary string.*

(2) *If S has an op-period p with shift i, then trace(S) "almost" has a period p, namely, trace$(S)[j]$ = trace$(S)[k]$ for any $j, k \in [\![1..n-1]\!]$ such that $j = k \pmod p$ and $j \ne i \pmod p$. (This is because both trace$(S)[j]$ and trace$(S)[k]$ equal the sign of the difference between the same positions of the shape of the op-period of S.)*

▶ **Example 6.** Consider the string 7 5 8 1 4 6 2 4 5. It has an op-period $(3, 1)$ with shape 2 3 1. The trace of this string is:

$$\texttt{- + - + + - + +}$$

The positions giving the remainder 1 modulo 3 are shown in gray; the sequence of the remaining positions is periodic.

It turns out that the existence of two coprime op-periods makes a string "almost" strictly monotone. One can use periodicity graphs [38, 39] to show the following result.

▶ **Theorem 7.** *Let S be a string of length n that has coprime op-periods p and q with shifts i and j, respectively, such that $n > p > q > 1$. Then:*

(a) *if $n > pq$, then S has a strictly monotone op-period pq;*

(b) *if $2p < n \le pq$ and the op-periods are synchronized, then S is 2-monotone;*

(c) *if $p+q < n \le 2p$ and the op-periods are synchronized, then (q, j) is a strictly monotone op-period of S;*

(d) *if $n > \max\{2p, p+2q\}$ and the op-periods are not synchronized, then S is strictly monotone;*

(e) *if $n > 2p$, the op-periods are not synchronized, and p is initial, then S is strictly monotone;*

(f) *if $p+q < n \le 2p$ and p is initial, then (q, j) is a strictly monotone op-period of S.*

3 Algorithmic Toolbox for Op-Model

For a string S of length n, we introduce a table op-PREF$[1..n]$ such that op-PREF$[i]$ is the length of the longest prefix of $S[i..n]$ that is equivalent to a prefix of S. It is a direct analogue of the PREF array used in standard string matching (see [20]) and can be computed similarly in $O(n)$ time using one of the standard encodings for the op-model that were used in [14, 18, 36].

▶ **Lemma 8.** *For a string of length n, the* op-PREF *table can be computed in $O(n)$ time.*

Let us mention an application of the op-PREF table that is used further in the algorithms. We denote by op-LPP$_p(S)$ ("longest op-periodic prefix") the length of the longest prefix of a string S having p as an initial op-period.

▶ **Lemma 9.** *For a string S of length n,* op-LPP$_p(S)$ *for a given p can be computed in $O(\text{op-LPP}_p(S)/p + 1)$ time after $O(n)$-time preprocessing.*

Proof. We start by computing the op-PREF table for S in $O(n)$ time. We assume that op-PREF$[n+1] = 0$. To compute op-LPP$_p(S)$, we iterate over positions $i = p+1, 2p+1, \dots$ and for each of them check if op-PREF$[i] \geq p$. If i_0 is the first position for which this condition is not satisfied (possibly because $i_0 > n-p+1$), we have op-LPP$_p(S) = i_0 + \text{op-PREF}[i_0] - 1$. Clearly, this procedure works in the desired time complexity. ◀

For a string S, we define a *longest common extension* query op-LCP(i,j) in the order-preserving model as the maximum $k \geq 0$ such that $S[i..i+k-1] \approx S[j..j+k-1]$. Symmetrically, op-LCS$(i,j)$ is the maximum $k \geq 0$ such that $S[i-k+1..i] \approx S[j-k+1..j]$.

Similarly as in the standard model [17], LCP-queries in the op-model can be answered using lowest common ancestor (LCA) queries in the op-suffix tree; see the following lemma.

▶ **Lemma 10.** *For a string of length n, after preprocessing in $O(n \log\log n)$ expected time or in $O(n \log^2 \log n / \log\log\log n)$ worst-case time one can answer* op-LCP*-queries in $O(1)$ time.*

The factor $S[i..i+2p-1]$ is called an order-preserving square (*op-square*) iff $S[i..i+p-1] \approx S[i+p..i+2p-1]$. For a string S of length n, we define the set

$$\textit{op-Squares}_p = \{i \in [\![1..n-2p+1]\!] \,:\, S[i..i+2p-1] \text{ is an op-square}\}.$$

Op-squares were first defined in [18] where an algorithm computing all the sets *op-Squares$_p$* for a string of length n in $O(n \log n + \sum_p |\textit{op-Squares}_p|)$ time was shown.

We say that an op-square $S[i..i+2p-1]$ is *right shiftable* if $S[i+1..i+2p]$ is an op-square and *right non-shiftable* otherwise. Similarly, we say that the op-square is *left shiftable* if $S[i-1..i+2p-2]$ is an op-square and *left non-shiftable* otherwise. Using the approach of [18], one can show the following lemma.

▶ **Lemma 11.** *All the (left and right) non-shiftable op-squares in a string of length n can be computed in $O(n \log n)$ time.*

4 Computing All Full and Initial Op-Periods

For a string S of length n, we define op-PREF$'[i]$ for $i = 0, \dots, n$ as:

$$\text{op-PREF}'[i] = \begin{cases} n & \text{if op-PREF}[i+1] = n-i \\ \text{op-PREF}[i+1] & \text{otherwise.} \end{cases}$$

Algorithm 1: Computing All Initial Op-Periods of S.

$T :=$ op-PREF$'$;
for $j := n$ **down to** 2 **do**
  foreach *prime divisor* q *of* j **do**
    $P[j/q] := \min(P[j/q], P[j])$;
for $p := 1$ **to** n **do**
  if $P[p] \geq p$ **then** p is an initial op-period;

Here we assume that op-PREF$[n+1] = 0$. In the computation of full and initial op-periods we heavily rely on this table according to the following obvious observation.

▶ **Observation 12.** *p is an initial op-period of a string S of length n if and only if* op-PREF$'[ip] \geq p$ *for all* $i = 1, \ldots, \lfloor n/p \rfloor$.

4.1 Computing Initial Op-Periods

Let us introduce an auxiliary array $P[0..n]$ such that:

$$P[p] = \min\{\text{op-PREF}'[ip] \,:\, i = 1, \ldots, \lfloor n/p \rfloor\}.$$

Straight from Observation 12 we have:

▶ **Observation 13.** *p is an initial period of S if and only if $P[p] \geq p$.*

The table T could be computed straight from definition in $O(n \log n)$ time. We improve this complexity to $O(n \log \log n)$ by employing Eratosthenes's sieve. The sieve computes, in particular, for each $j = 1, \ldots, n$ a list of all distinct prime divisors of j. We use these divisors to compute the table via dynamic programming in a right-to-left scan, as shown in Algorithm 1.

▶ **Theorem 14.** *All initial op-periods of a string of length n can be computed in $O(n \log \log n)$ time.*

Proof. By Lemma 8, the op-PREF table for the string—hence, the op-PREF$'$ table—can be computed in $O(n)$ time. Then we use Algorithm 1. Each prime number $q \leq n$ has at most $\frac{n}{q}$ multiples below n. Therefore, the complexity of Eratosthenes's sieve and the number of updates on the table T in the algorithm is $\sum_{q \in Primes, q \leq n} \frac{n}{q} = O(n \log \log n)$; see [1]. ◀

4.2 Computing Full Op-Periods

Let us recall the following auxiliary data structure for efficient gcd-computations that was developed in [34]. We will only need a special case of this data structure to answer queries for $\gcd(x, n)$.

▶ **Fact 15** (Theorem 4 in [34]). *After $O(n)$-time preprocessing, given any $x, y \in \{1, \ldots, n\}$, the value $\gcd(x, y)$ can be computed in constant time.*

Let $Div(i)$ denote the set of all positive divisors of i. In the case of full op-periods we only need to compute $P[p]$ for $p \in Div(n)$. As in Algorithm 1, we start with $T =$ op-PREF$'$. Then we perform a preprocessing phase that shifts the information stored in the array from

Algorithm 2: Computing All Full Op-Periods of S.

```
T := op-PREF′;
for i := 1 to n do
    k := gcd(i, n);
    P[k] := min(P[k], P[i]);
foreach i ∈ Div(n) in decreasing order do
    foreach d ∈ Div(i) do
        P[d] := min(P[d], P[i]);
foreach p ∈ Div(n) do
    if P[p] ≥ p then p is a full op-period;
```

indices $i \notin Div(n)$ to indices $\gcd(i,n) \in Div(n)$. It is based on the fact that for $d \in Div(n)$, $d \mid i$ if and only if $d \mid \gcd(i,n)$. Finally, we perform right-to-left processing as in Algorithm 1. However, this time we can afford to iterate over all divisors of elements from $Div(n)$. Thus we arrive at the pseudocode of Algorithm 2.

▶ **Theorem 16.** *All full op-periods of a string of length n can be computed in $O(n)$ time.*

Proof. We apply Algorithm 2. The complexity of the first for-loop is $O(n)$ by Fact 15. The second for-loop works in $O(n)$ time as the sizes of the sets $Div(n)$, $Div(i)$ are $O(\sqrt{n})$ and the elements of these sets can be enumerated in $O(\sqrt{n})$ time as well. ◀

5 Computing Smallest Non-Trivial Initial Op-Period

If a string is not strictly monotone itself, it has $O(n)$ such op-periods and they can all be computed in $O(n)$ time. We use this as an auxiliary routine in the computation of the smallest initial op-period that is greater than 1.

▶ **Theorem 17.** *If a string of length n is not strictly monotone, all of its strictly monotone op-periods can be computed in $O(n)$ time.*

Let us start with the following simple property.

▶ **Lemma 18.** *The shape of the smallest non-trivial initial op-period of a string has no shorter non-trivial full op-period.*

Proof. A full op-period of the initial op-period of a string S is an initial op-period of S. ◀

Now we can state a property of initial op-periods, implied by Theorem 7, that is the basis of the algorithm.

▶ **Lemma 19.** *If a string of length n has initial op-periods $p > q > 1$ such that $p + q < n$ and $\gcd(p,q) = 1$, then q is strictly monotone.*

Proof. Let us consider three cases. If $n > pq$, then by Theorem 7(a), both p and q are strictly monotone. If $2p < n \le pq$, then Theorem 7(e) implies that $S[1..pq-1]$ is strictly monotone, hence p and q are strictly monotone as well. Finally, if $p + q < n \le 2p$, we have that q is strictly monotone by Theorem 7(f). ◀

Algorithm 3: Computing the Smallest Non-Trivial Initial Op-Period of S.

if *S has a non-trivial strictly monotone op-period* **then**
    return *smallest such op-period*; ▷ Theorem 17
$p :=$ the length of the longest monotone prefix of S plus 1;
while $p \le n$ **do**
    $k :=$ op-$\mathrm{LPP}_p(S)$;
    if $k = n$ **then return** p;
    $p := \max(p+1,\ k-p-1)$;
return $\min(p_{mon}, n)$;

▶ **Theorem 20.** *The smallest initial op-period $p > 1$ of a string S of length n can be computed in $O(n)$ time.*

Proof. We follow the lines of Algorithm 3. If S is not strictly monotone itself, we can compute the smallest non-trivial strictly monotone initial op-period of S using Theorem 17. Otherwise, the smallest such op-period is 2. If S has a non-trivial strictly monotone initial op-period and the smallest such op-period is $q > 1$, then none of $2, \dots, q-1$ is an initial op-period of S. Hence, we can safely return q.

Let us now focus on the correctness of the while-loop. The invariant is that there is no initial op-period of S that is smaller than p. If the value of $k =$ op-$\mathrm{LPP}_p(S)$ equals n, then p is an initial op-period of S and we can safely return it. Otherwise, we can advance p by 1. There is also no smallest initial op-period p' such that $p < p' < k-p-1$. Indeed, Lemma 19 would imply that p is strictly monotone if $\gcd(p,p') = 1$ (which is impossible due to the initial selection of p) and Theorem 3 would imply an initial op-period of $S[1..p']$ that is smaller than p' and divides p' if $\gcd(p,p') > 1$ (which is impossible due to Lemma 18). This justifies the way p is increased.

Now let us consider the time complexity of the algorithm. The algorithm for strictly monotone op-periods of Theorem 17 works in $O(n)$ time. By Lemma 9, k can be computed in $O(k/p+1)$ time. If $k \le 3p$, this is $O(1)$. Otherwise, p at least doubles; let p' be the new value of p. Then $O(k/p+1) = O((p+p'-1)/p+1) = O(p'+1)$. The case that p doubles can take place at most $O(\log n)$ times and the total sum of p' over such cases is $O(n)$. ◀

6 Computing All Op-Periods

An *interval representation* of a set X of integers is $X = [\![i_1..j_1]\!] \cup [\![i_2..j_2]\!] \cup \dots \cup [\![i_k..j_k]\!]$ where $j_1+1 < i_2, \dots, j_{k-1}+1 < i_k$; k is called the *size* of the representation.

Our goal is to compute a *compact representation* of all the op-periods of a string that contains, for each op-period p, an interval representation of the set Shifts_p.

For an integer set X, by $X \bmod p$ we denote the set $\{x \bmod p : x \in X\}$. The following technical lemma provides efficient operations on interval representations of sets.

▶ **Lemma 21.**

(a) *Assume that X and Y are two sets with interval representations of sizes x and y, respectively. Then the interval representation of the set $X \cap Y$ can be computed in $O(x+y)$ time.*

(b) *Assume that $X_1, \dots, X_k \subseteq [\![0..n]\!]$ are sets with interval representations of sizes $x_1, \dots, x_k$ and $p_1, \dots, p_k$ be positive integers. Then the interval representations of all the sets $X_1 \bmod p_1, \dots, X_k \bmod p_k$ can be computed in $O(x_1 + \dots + x_k + k + n)$ time.*

Algorithm 4: Computing a Compact Representation of All Op-Periods.

Compute $\textit{op-Squares}_p$ for all $p = 1, \ldots, n$; ▷ Lemma 22
for $p := 1$ **to** n **do**
  $\mathcal{N}_p := [\![1..n-2p+1]\!] \setminus \textit{op-Squares}_p$;
  $k :=$ op-LCP$(1, p+1)$; $\ell :=$ op-LCS$(n, n-p)$;
  if $k = n - p$ **then** $\mathcal{B}_p := \mathcal{C}_p := [\![1..n]\!]$;
  else $\mathcal{B}_p := [\![1..k]\!]$; $\mathcal{C}_p := [\![n-\ell+1..n]\!]$;
for $p := 1$ **to** n *simultaneously* **do**
  $\mathcal{N}_p := \{(x-1) \bmod p : x \in \mathcal{N}_p\}$; $\mathcal{B}_p := \mathcal{B}_p \bmod p$; $\mathcal{C}_p := \mathcal{C}_p \bmod p$; ▷ Lemma 21(b)
$\textit{Shifts}_1 := [\![0]\!]$;
for $p := 2$ **to** n **do**
  $\mathcal{A}_p := [\![0..p-1]\!] \setminus \mathcal{N}_p$;
  $\textit{Shifts}_p := \mathcal{A}_p \cap \mathcal{B}_p \cap \mathcal{C}_p$; ▷ Lemma 21(a)
return $\textit{Shifts}_p$ *for* $p = 1, \ldots, n$;

► **Lemma 22.** *For a string of length n, interval representations of the sets $\textit{op-Squares}_p$ for all $1 \le p \le n/2$ can be computed in $O(n \log n)$ time.*

Proof. Let us define the following two auxiliary sets.

$$\mathcal{L}_p = \{i \in [\![1..n-2p+1]\!] \,:\, S[i..i+2p-1] \text{ is a left non-shiftable op-square}\}$$
$$\mathcal{R}_p = \{i \in [\![1..n-2p+1]\!] \,:\, S[i..i+2p-1] \text{ is a right non-shiftable op-square}\}.$$

By Lemma 11, all the sets $\mathcal{L}_p$ and $\mathcal{R}_p$ can be computed in $O(n \log n)$ time. In particular, $\sum_p |\mathcal{L}_p| = O(n \log n)$.

Let us note that, for each p, $|\mathcal{L}_p| = |\mathcal{R}_p|$. Thus let $\mathcal{L}_p = \{\ell_1, \ldots, \ell_k\}$ and $\mathcal{R}_p = \{r_1, \ldots, r_k\}$. The interval representation of the set $\textit{op-Squares}_p$ is $[\![\ell_1..r_1]\!] \cup \cdots \cup [\![\ell_k..r_k]\!]$. Clearly, it can be computed in $O(|\mathcal{L}_p|)$ time. ◄

We will use the following characterization of op-periods.

► **Observation 23.** *p is an op-period of S with shift i if and only if all the following conditions hold:*

(A) *$S[i+1+kp..i+(k+2)p]$ is an op-square for every $0 \le k \le (n-2p-i)/p$,*
(B) op-LCP$(1, p+1) \ge \min(i, n-p)$,
(C) op-LCS$(n, n-p) \ge \min((n-i) \bmod p, n-p)$.

► **Theorem 24.** *A representation of size $O(n \log n)$ of all the op-periods of a string of length n can be computed in $O(n \log n)$ time.*

Proof. We use Algorithm 4. The sets $\mathcal{A}_p$, $\mathcal{B}_p$, and $\mathcal{C}_p$ describe the sets of shifts i that satisfy conditions (A), (B), and (C) from Observation 23, respectively.

A crucial role is played by the set $\mathcal{N}_p$ of all positions which are *not* the beginnings of op-squares of length $2p$. It is computed as a complement of the set $\textit{op-Squares}_p$.

Operations "mod" on sets are performed simultaneously using Lemma 21(b). All sets $\mathcal{A}_p$, $\mathcal{B}_p$, $\mathcal{C}_p$ have $O(n \log n)$-sized representations. This guarantees $O(n \log n)$ time. ◄

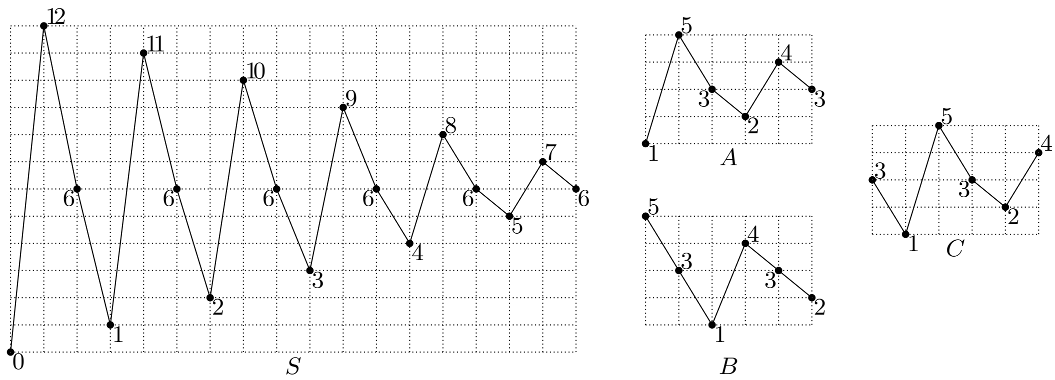

Figure 3 A string $S = 0\,12\,6\,1\,11\,6\,2\,10\,6\,3\,9\,6\,4\,8\,6\,5\,7\,6$ is graphically illustrated above (the ith point has coordinates $(i, S[i])$). We have $SH_6 = ABCABCABCA$, where $A = 1\,5\,3\,2\,4\,3$, $B = 5\,3\,1\,4\,3\,2$, and $C = 3\,1\,5\,3\,2\,4$. The shortest period of SH_6 is 3. Hence, 6 is a sliding op-period of S. Moreover, Lemma 27b implies that 3 is a period of SH_3, hence a sliding op-period of S.

7 Computing Sliding Op-Periods

For a string S of length n, we define a family of strings $SH_1, \dots, SH_n$ such that $SH_k[i] = \mathit{shape}(S[i..i+k-1])$ for $1 \le i \le n-k+1$. Note that the characters of the strings are shapes. Moreover, the total length of strings SH_k is quadratic in n, so we will not compute those strings explicitly. Instead, we use the following observation to test if two symbols are equal.

▶ **Observation 25.** *$SH_k[i] = SH_k[i']$ if and only if* op-LCP$(i, i') \ge k$.

Sliding op-periods admit an elegant characterization based on SH_k; see Figure 3.

▶ **Lemma 26.** *An integer p, $1 \le p \le n$, is a sliding op-period of S if and only if $p \le \frac{1}{2}n$ and p is a period of SH_p, or $p > \frac{1}{2}n$ and $S[1..n-p] \approx S[p+1..n]$.*

For a string X, we denote the shortest period of X by $\mathsf{per}(X)$.

▶ **Lemma 27.** *Suppose that $p = \mathsf{per}(SH_k[1..\ell]) < \ell$. Then*
(a) *p is also a period of $SH_{k'}[1..\ell+k-k']$ for $1 \le k' \le k$,*
(b) *$q = \mathsf{per}(SH_k[1..\ell+1])$ satisfies $p = q$ or $p + q > \ell$.*

We introduce a two-dimensional table PER, where:

$$PER[k, \ell] = \mathsf{per}(SH_k[1..\ell]) \text{ if } \mathsf{per}(SH_k[1..\ell]) \le \frac{1}{3}\ell, \text{ and } PER[k, \ell] = \perp \text{(undefined) otherwise.}$$

The size of PER is quadratic in n. However, Algorithm 5 computes PER column after column, keeping only the current column $P = PER[\cdot, \ell]$. The total number of differences between consecutive columns is linear. Hence, any requested $O(n)$ values $PER[k, \ell]$ can be computed in $O(n)$ time. We also use an analogous table PER^R for the reverse string S^R.

▶ **Lemma 28.** *Algorithm 5 is correct, that is, it satisfies the invariant.*

Proof. First, observe that the invariant is satisfied after the first iteration. This is because $\mathsf{per}(SH_k[1..1]) = 1$ for each k and the initial values are not changed during this iteration.

Thus, our task is to prove that the invariant is preserved after each subsequent ℓth iteration. Let $t = \min\{k : PER[k, \ell-1] = \perp\}$ and $t' = \min\{k : PER[k, \ell] = \perp\}$.

Algorithm 5: Computation of $PER[\cdot, \ell]$ from $PER[\cdot, \ell - 1]$.

$P[1..n] := [\bot, \ldots, \bot]$; $t := 1$; $\ell' := 3$;
for $\ell := 1$ **to** n **do**
  if $t > 1$ **and** $SH_{t-1}[\ell] \neq SH_{t-1}[\ell - P[t-1]]$ **then**
    $t := t - 1$; $P[t] := \bot$; $\ell' := 2\ell$;
  if $\ell \geq \ell'$ **then**
    while $\mathsf{per}(SH_t[1..\ell]) = \frac{1}{3}\ell$ **do**
      $P[t] := \frac{1}{3}\ell$; $t := t + 1$; $\ell' := 2\ell$;
  ▷ Invariant: $P[k] = PER[k, \ell]$, $t = \min\{k : P[k] = \bot\}$, and $\mathsf{per}(SH_t[1..\ell]) \geq \frac{1}{3}\ell'$.

Algorithm 6: Computing the sliding op-periods $p \leq \frac{1}{2}n$.

$p := 1$;
while $p \leq \frac{1}{2}n$ **do**
  if $(q := PER[p, n - 2p + 1]) = PER^R[p, n - 2p + 1] \neq \bot$ **then**
    if *p is a period of $SH_p[1..p + q]$* **then** report p;
    $p := \min\{p' > p : p'$ is a period of $SH_p[1..p + 2q]\}$
  else if $PER[p, \lceil \frac{3}{4}(n - 2p + 1)\rceil] = PER^R[p, \lceil \frac{3}{4}(n - 2p + 1)\rceil] \neq \bot$ **then** $p := p + 1$;
  else
    if *p is a period of SH_p* **then** report p;
    $p := \min\{p' > p : p'$ is a period of $SH_p\}$;

First, we consider the values $PER[k, \ell]$ for $k < t$. For this, we assume $t > 1$ and denote $p = PER[t - 1, \ell - 1]$. Since p is a period of $SH_{t-1}[1..\ell - 1]$, Lemma 27a yields that p is also a period of $SH_k[1..\ell]$ for $k < t - 1$. We apply Lemma 27b for $p' = \mathsf{per}(SH_k[1..\ell - 1])$. Since $p' + p \leq \ell - 1$, we conclude that $p' = \mathsf{per}(SH_k[1..\ell])$, i.e., $PER[k, \ell - 1] = p' = PER[k, \ell]$. Now, we consider the value $PER[t - 1, \ell]$. Lemma 27b, applied for $p = \mathsf{per}(SH_{t-1}[1..\ell - 1])$ and $q = \mathsf{per}(SH_{t-1}[1..\ell])$, yields $p = q$ or $p + q \geq \ell$. To verify the first case, we check whether $SH_{t-1}[\ell] = SH_{t-1}[\ell - p]$. In the second case, we conclude that $q \geq \frac{2}{3}\ell$, so $PER[t - 1, \ell] = \bot$ (and $\ell' := 2\ell$ is also set correctly).

Next, we consider the values $PER[k, \ell]$ for $k \geq t$. Since $PER[k, \ell - 1] = \bot$, we have $PER[k, \ell] = \bot$ or $PER[k, \ell] = \frac{1}{3}\ell$. More precisely, $PER[k, \ell] = \bot$ for $k \geq t'$ and $PER[k, \ell] = \frac{1}{3}\ell$ for $t \leq k < t'$. Thus, we check if $\mathsf{per}(SH_k[1..\ell]) = \frac{1}{3}\ell$ for subsequent values $k \geq t$. Since $\mathsf{per}(SH_t[1..\ell]) \geq \frac{1}{3}\ell'$, no verification is needed if $\ell < \ell'$. To complete the proof, we need to show that the update $\ell' := 2\ell$ is valid if $t' > t$. For a proof by contradiction suppose that $r := \mathsf{per}(SH_{t'}[1..\ell]) < \frac{2}{3}\ell$. By Lemma 27a, r is a period of $SH_t[1..\ell]$. Since $r + \frac{1}{3}\ell \leq \ell$, Periodicity Lemma yields $\frac{1}{3}\ell \mid r$, and thus $r = \frac{1}{3}\ell$, which contradicts the definition of t'. ◀

▶ **Lemma 29.** *Algorithm 5 can be implemented in time $O(n)$ plus the time to answer $O(n)$* op-LCP *queries in S.*

▶ **Lemma 30.** *Algorithm 6 is correct, that is, it reports all sliding op-periods $p \leq \frac{1}{2}n$ of S.*

Proof. Let p_i be the value of p at the beginning of the ith iteration of the while-loop and let $\ell_i = n - 2p_i + 1$. We shall prove that p_i is reported if and only if it is a sliding op-period and that there is no sliding op-period strictly between p_i and p_{i+1}.

First, suppose that $q = \mathsf{per}(SH_{p_i}[1..\ell_i]) = \mathsf{per}(SH_{p_i}[p_i + 1..p_i + \ell_i]) \leq \frac{1}{3}\ell_i$, i.e., we are in the first branch. If $SH_{p_i}[1..q] = SH_{p_i}[p_i + 1..p_i + q]$, then we must have $SH_{p_i}[1..\ell_i] =$

$SH_{p_i}[p_i+1..p_i+\ell_i]$, i.e., p_i is a period of $SH_{p_i} = SH_{p_i}[1..p_i+\ell_i]$ and p_i is a sliding op-period due to Lemma 26. Moreover, any sliding op-period $p' > p_i$ must be a period of SH_{p_i} (and, in particular, of $SH_{p_i}[1..p_i+2q]$) due to Lemma 27a. Consequently, $p' \geq p_{i+1}$, as claimed.

In the second branch we only need to prove that $SH_{p_i}[1..\ell_i] \neq SH_{p_i}[p_i+1..p_i+\ell_i]$. For a proof by contradiction, suppose that we have an equality. The condition from Line 6 means that the length-$\lceil\frac{3}{4}\ell_i\rceil$ prefix and suffix of $SH_{p_i}[1..\ell_i] = SH_{p_i}[p_i+1..p_i+\ell_i]$ has the common shortest period $q \leq \frac{1}{3}\lceil\frac{3}{4}\ell_i\rceil \leq \lceil\frac{1}{4}\ell_i\rceil$. The prefix and the suffix overlap by at least $\lceil\frac{1}{2}\ell_i\rceil$ characters, so we actually have $q = \mathsf{per}(SH_{p_i}[1..\ell_i]) = \mathsf{per}(SH_{p_i}[p_i+1..p_i+\ell_i])$. Hence, in that case we would be in the first branch.

Finally, in the third branch we directly use Lemma 26 to check if p_i is a sliding op-period. Moreover, if $p' > p_i$ is also a sliding op-period, then p' is a period of SH_{p_i}, i.e., $p' \geq p_{i+1}$. ◀

Let us observe that $PER[k,\ell]$ and $PER^R[k,\ell]$ is used in Algorithm 6 only for $\ell = n-2k+1$ or $\ell = \left\lceil\frac{3}{4}(n-2k+1)\right\rceil$. These $O(n)$ values can be computed in $O(n)$ time using Algorithm 5. In [29] we show the following lemma.

▶ **Lemma 31.** *Algorithm 6 can be implemented in time $O(n)$ plus the time to answer $O(n)$* op-LCP *and* op-LCS *queries in S.*

▶ **Theorem 32.** *All sliding op-periods of a string of length n can be computed in $O(n)$ space and $O(n\log\log n)$ expected time or $O(n\log^2\log n/\log\log\log n)$ worst-case time.*

Proof. First, we apply Lemma 10 so that op-LCP and op-LCS queries can be answered in $O(1)$ time. Next, we run Algorithm 6 to report sliding op-periods $p \leq \frac{1}{2}n$. Then, we iterate over $p > \frac{1}{2}n$ and report p if op-LCP$(1, p+1) = n-p$. Correctness follows from Lemmas 30 and 26. The overall time is $O(n)$ (Lemma 31) plus the preprocessing time of Lemma 10. ◀

References

1 Tom M. Apostol. *Introduction to Analytic Number Theory.* Undergraduate Texts in Mathematics, Springer, 1976.

2 Alberto Apostolico and Raffaele Giancarlo. Periodicity and repetitions in parameterized strings. *Discrete Applied Mathematics*, 156(9):1389–1398, 2008. doi:10.1016/j.dam.2006.11.017.

3 Djamal Belazzougui, Adeline Pierrot, Mathieu Raffinot, and Stéphane Vialette. Single and multiple consecutive permutation motif search. In Leizhen Cai, Siu-Wing Cheng, and Tak Wah Lam, editors, *Algorithms and Computation - 24th International Symposium, ISAAC 2013, Hong Kong, China, December 16-18, 2013, Proceedings*, volume 8283 of *Lecture Notes in Computer Science*, pages 66–77. Springer, 2013. doi:10.1007/978-3-642-45030-3_7.

4 Jean Berstel and Luc Boasson. Partial words and a theorem of fine and wilf. *Theor. Comput. Sci.*, 218(1):135–141, 1999. doi:10.1016/S0304-3975(98)00255-2.

5 Francine Blanchet-Sadri, Deepak Bal, and Gautam Sisodia. Graph connectivity, partial words, and a theorem of fine and wilf. *Inf. Comput.*, 206(5):676–693, 2008. doi:10.1016/j.ic.2007.11.007.

6 Francine Blanchet-Sadri and Robert A. Hegstrom. Partial words and a theorem of fine and wilf revisited. *Theor. Comput. Sci.*, 270(1-2):401–419, 2002. doi:10.1016/S0304-3975(00)00407-2.

7 Francine Blanchet-Sadri, Sean Simmons, Amelia Tebbe, and Amy Veprauskas. Abelian periods, partial words, and an extension of a theorem of fine and wilf. *RAIRO - Theor. Inf. and Applic.*, 47(3):215–234, 2013. doi:10.1051/ita/2013034.

8 Domenico Cantone, Simone Faro, and M. Oguzhan Külekci. An efficient skip-search approach to the order-preserving pattern matching problem. In Jan Holub and Jan Zdárek, editors, *Proceedings of the Prague Stringology Conference 2015, Prague, Czech Republic, August 24-26, 2015*, pages 22–35. Department of Theoretical Computer Science, Faculty of Information Technology, Czech Technical University in Prague, 2015. URL: `http://www.stringology.org/event/2015/p04.html`.

9 Maria Gabriella Castelli, Filippo Mignosi, and Antonio Restivo. Fine and wilf's theorem for three periods and a generalization of sturmian words. *Theor. Comput. Sci.*, 218(1):83–94, 1999. `doi:10.1016/S0304-3975(98)00251-5`.

10 Tamanna Chhabra, Simone Faro, M. Oguzhan Külekci, and Jorma Tarhio. Engineering order-preserving pattern matching with SIMD parallelism. *Softw., Pract. Exper.*, 47(5):731–739, 2017. `doi:10.1002/spe.2433`.

11 Tamanna Chhabra, Emanuele Giaquinta, and Jorma Tarhio. Filtration algorithms for approximate order-preserving matching. In Costas S. Iliopoulos, Simon J. Puglisi, and Emine Yilmaz, editors, *String Processing and Information Retrieval - 22nd International Symposium, SPIRE 2015, London, UK, September 1-4, 2015, Proceedings*, volume 9309 of *Lecture Notes in Computer Science*, pages 177–187. Springer, 2015. `doi:10.1007/978-3-319-23826-5_18`.

12 Tamanna Chhabra, M. Oguzhan Külekci, and Jorma Tarhio. Alternative algorithms for order-preserving matching. In Jan Holub and Jan Zdárek, editors, *Proceedings of the Prague Stringology Conference 2015, Prague, Czech Republic, August 24-26, 2015*, pages 36–46. Department of Theoretical Computer Science, Faculty of Information Technology, Czech Technical University in Prague, 2015. URL: `http://www.stringology.org/event/2015/p05.html`.

13 Tamanna Chhabra and Jorma Tarhio. A filtration method for order-preserving matching. *Inf. Process. Lett.*, 116(2):71–74, 2016. `doi:10.1016/j.ipl.2015.10.005`.

14 Sukhyeun Cho, Joong Chae Na, Kunsoo Park, and Jeong Seop Sim. A fast algorithm for order-preserving pattern matching. *Inf. Process. Lett.*, 115(2):397–402, 2015. `doi:10.1016/j.ipl.2014.10.018`.

15 Sorin Constantinescu and Lucian Ilie. Fine and Wilf's theorem for Abelian periods. *Bulletin of the EATCS*, 89:167–170, 2006.

16 Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, 3rd Edition.* MIT Press, 2009. URL: `http://mitpress.mit.edu/books/introduction-algorithms`.

17 Maxime Crochemore, Christophe Hancart, and Thierry Lecroq. *Algorithms on strings.* Cambridge University Press, 2007.

18 Maxime Crochemore, Costas S. Iliopoulos, Tomasz Kociumaka, Marcin Kubica, Alessio Langiu, Solon P. Pissis, Jakub Radoszewski, Wojciech Rytter, and Tomasz Walen. Order-preserving indexing. *Theor. Comput. Sci.*, 638:122–135, 2016. `doi:10.1016/j.tcs.2015.06.050`.

19 Maxime Crochemore, Costas S. Iliopoulos, Tomasz Kociumaka, Marcin Kubica, Jakub Pachocki, Jakub Radoszewski, Wojciech Rytter, Wojciech Tyczynski, and Tomasz Walen. A note on efficient computation of all abelian periods in a string. *Inf. Process. Lett.*, 113(3):74–77, 2013. `doi:10.1016/j.ipl.2012.11.001`.

20 Maxime Crochemore and Wojciech Rytter. *Jewels of Stringology.* World Scientific, 2003.

21 Gianni Decaroli, Travis Gagie, and Giovanni Manzini. A compact index for order-preserving pattern matching. In Ali Bilgin, Michael W. Marcellin, Joan Serra-Sagristà, and James A. Storer, editors, *2017 Data Compression Conference, DCC 2017, Snowbird, UT, USA, April 4-7, 2017*, pages 72–81. IEEE, 2017. `doi:10.1109/DCC.2017.35`.

22 Sergi Elizalde and Marc Noy. Consecutive patterns in permutations. *Advances in Applied Mathematics*, 30(1):110–125, 2003. doi:10.1016/S0196-8858(02)00527-4.

23 Simone Faro and M. Oguzhan Külekci. Efficient algorithms for the order preserving pattern matching problem. In Riccardo Dondi, Guillaume Fertin, and Giancarlo Mauri, editors, *Algorithmic Aspects in Information and Management - 11th International Conference, AAIM 2016, Bergamo, Italy, July 18-20, 2016, Proceedings*, volume 9778 of *Lecture Notes in Computer Science*, pages 185–196. Springer, 2016. doi:10.1007/978-3-319-41168-2_16.

24 Gabriele Fici, Thierry Lecroq, Arnaud Lefebvre, and Élise Prieur-Gaston. Algorithms for computing abelian periods of words. *Discrete Applied Mathematics*, 163:287–297, 2014. doi:10.1016/j.dam.2013.08.021.

25 Gabriele Fici, Thierry Lecroq, Arnaud Lefebvre, Élise Prieur-Gaston, and William F. Smyth. A note on easy and efficient computation of full abelian periods of a word. *Discrete Applied Mathematics*, 212:88–95, 2016. doi:10.1016/j.dam.2015.09.024.

26 Nathan J. Fine and Herbert S. Wilf. Uniqueness theorems for periodic functions. *Proc. Amer. Math. Soc.*, 16:109–114, 1965.

27 Travis Gagie, Giovanni Manzini, and Rossano Venturini. An encoding for order-preserving matching. In Kirk Pruhs and Christian Sohler, editors, *25th Annual European Symposium on Algorithms, ESA 2017, September 4-6, 2017, Vienna, Austria*, volume 87 of *LIPIcs*, pages 38:1–38:15. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017. doi:10.4230/LIPIcs.ESA.2017.38.

28 Pawel Gawrychowski and Przemyslaw Uznanski. Order-preserving pattern matching with k mismatches. *Theor. Comput. Sci.*, 638:136–144, 2016. doi:10.1016/j.tcs.2015.08.022.

29 Garance Gourdel, Tomasz Kociumaka, Jakub Radoszewski, Wojciech Rytter, Arseny Shur, and Tomasz Waleń. String periods in the order-preserving model. ArXiv preprint. URL: https://arxiv.org/abs/1801.01404.

30 Lidia A. Idiatulina and Arseny M. Shur. Periodic partial words and random bipartite graphs. *Fundam. Inform.*, 132(1):15–31, 2014. doi:10.3233/FI-2014-1030.

31 Jacques Justin. On a paper by castelli, mignosi, restivo. *ITA*, 34(5):373–377, 2000. doi:10.1051/ita:2000122.

32 Jinil Kim, Peter Eades, Rudolf Fleischer, Seok-Hee Hong, Costas S. Iliopoulos, Kunsoo Park, Simon J. Puglisi, and Takeshi Tokuyama. Order-preserving matching. *Theor. Comput. Sci.*, 525:68–79, 2014. doi:10.1016/j.tcs.2013.10.006.

33 Tomasz Kociumaka, Jakub Radoszewski, and Wojciech Rytter. Fast algorithms for abelian periods in words and greatest common divisor queries. In Natacha Portier and Thomas Wilke, editors, *30th International Symposium on Theoretical Aspects of Computer Science, STACS 2013, February 27 - March 2, 2013, Kiel, Germany*, volume 20 of *LIPIcs*, pages 245–256. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2013. doi:10.4230/LIPIcs.STACS.2013.245.

34 Tomasz Kociumaka, Jakub Radoszewski, and Wojciech Rytter. Fast algorithms for abelian periods in words and greatest common divisor queries. *J. Comput. Syst. Sci.*, 84:205–218, 2017. doi:10.1016/j.jcss.2016.09.003.

35 Tomasz Kociumaka, Jakub Radoszewski, and Bartlomiej Wisniewski. Subquadratic-time algorithms for abelian stringology problems. In Ilias S. Kotsireas, Siegfried M. Rump, and Chee K. Yap, editors, *Mathematical Aspects of Computer and Information Sciences - 6th International Conference, MACIS 2015, Berlin, Germany, November 11-13, 2015, Revised Selected Papers*, volume 9582 of *Lecture Notes in Computer Science*, pages 320–334. Springer, 2015. doi:10.1007/978-3-319-32859-1_27.

36 Marcin Kubica, Tomasz Kulczynski, Jakub Radoszewski, Wojciech Rytter, and Tomasz Walen. A linear time algorithm for consecutive permutation pattern matching. *Inf. Process. Lett.*, 113(12):430–433, 2013. doi:10.1016/j.ipl.2013.03.015.

37 Yoshiaki Matsuoka, Takahiro Aoki, Shunsuke Inenaga, Hideo Bannai, and Masayuki Takeda. Generalized pattern matching and periodicity under substring consistent equivalence relations. *Theor. Comput. Sci.*, 656:225–233, 2016. doi:10.1016/j.tcs.2016.02.017.

38 Arseny M. Shur and Yulia V. Gamzova. Partial words and the interaction property of periods. *Izvestiya: Mathematics*, 68:405–428, 2004. URL: http://stacks.iop.org/1064-5632/68/i=2/a=A09.

39 Arseny M. Shur and Yulia V. Konovalova. On the periods of partial words. In Jirí Sgall, Ales Pultr, and Petr Kolman, editors, *Mathematical Foundations of Computer Science 2001, 26th International Symposium, MFCS 2001 Marianske Lazne, Czech Republic, August 27-31, 2001, Proceedings*, volume 2136 of *Lecture Notes in Computer Science*, pages 657–665. Springer, 2001. doi:10.1007/3-540-44683-4_57.

40 Robert Tijdeman and Luca Zamboni. Fine and Wilf words for any periods. *Indag. Math.*, 14(1):135–147, 2003. doi:10.1016/S0019-3577(03)90076-0.