Incremental algorithms for computing the set of period sets

Eric Rivals

Incremental algorithms for computing the set of period sets

Eric Rivals

LIRMM, Université Montpellier, CNRS, Montpellier, France

Abstract

Overlaps between strings are crucial in many areas of computer science, such as bioinformatics, code design, and stringology. A self overlapping string is characterized by its periods and borders. A period of a string u is the starting position of a suffix of u that is also a prefix u, and such a suffix is called a border. Each word of length, say $n > 0$, has a set of periods, but not all combinations of integers are sets of periods. The question we address is how to compute the set, denoted Γ_n, of all period sets of strings of length n. Computing the period set for all possible words of length n is clearly prohibitive. The cardinality of Γ_n is exponential in n. One dynamic programming algorithm exists for enumerating Γ_n, but it suffers from an expensive space complexity. After stating some combinatorial properties of period sets, we present a novel algorithm that computes Γ_n from Γ_{n-1}, for any length $n > 1$. The period set of a string u is a key information for computing the absence probability of u in random texts. Hence, computing Γ_n is useful for assessing the significance of word statistics, such as the number of missing k-mers in a random text, or the number of shared k-mers between two random texts. Besides applications, investigating Γ_n is interesting per se as it unveils combinatorial properties of string overlaps.

2012 ACM Subject Classification Mathematics of computing → Discrete mathematics

Keywords and phrases autocorrelation, string, period, combinatorics, periodicity

Digital Object Identifier 10.4230/LIPIcs.???.2024.

Funding E. Rivals is supported by the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 956229.

1 Introduction

Considering finite words or strings over a finite alphabet, we say that a word u overlaps a word v if a suffix of u of length, say i, equals a prefix of v of the same length. A pair of words u, v can have several overlaps of different lengths. For instance, over the binary alphabet $\{a, b\}$, consider the words $u := ababba$ and $v := abbabb$: u overlaps v with the suffix-prefix $abba$, and with a. It appears that the longest overlap contains all other overlaps: to find all overlaps from u to v, it suffices to study the overlaps of $abba$ with itself. This is true in general for any pair of words.

For a word u, a suffix that equals a prefix of u is called a border, and the length of u minus the length of a border, is called a period. Computing all self-overlaps of a word u is computing all its borders or all its periods, which can be done in linear time (see [28]). For instance the word *abracadabra* of length $n = 11$ has the following set of periods $\{0, 7, 10\}$ (zero being the trivial period - the whole word matches itself). This problem and variants of it have been widely studied, since it is useful in the design of pattern matching algorithms (like [12]).

The reader can easily convince her/himself that distinct words of the same length can share the same set of periods, even if one forbids a permutation of the alphabet. For a word u, let us denote by $P(u)$ its period set (which we abbreviate by PS). In this work, we investigate algorithms to enumerate all possible period sets for any words of a given length n. This set is denoted Γ_n for $n > 0$ and is non trivial if the alphabet contains at least two symbols. Brute force enumeration can consider all possible words of length n and compute their period set, but this approach obviously becomes computationally unaffordable for $n > 30$.

Interest in Γ_n sparkled mostly in the 80's, when researchers started to evaluate the average behavior of pattern matching algorithms, or that of filtering strategies for sequence alignment, text comparisons or clustering. A powerful filtering when comparing two texts, is to list their k-mers, for appropriate values of k, and then compute e.g. a Jaccard distance between their k-mer spectrum, to see whether the two texts are similar enough to warrant a costly alignment procedure [31].

In a different area, testing Pseudo-Random Number Generators can also be translated into a question on vocabulary statistics. Indeed in truly random real numbers written as sequence of digits, all substrings of a given length, say k, should ideally have an almost equal number of occurrences. In other words, for any substring the number of its occurrences in the sequence should not significantly deviate from a theoretical expectation. Empirical tests, named *Monkey Tests*, were developed for such generators [17, 20, 14]. It turns out that the absence probability of a word/string in a random text is essentially controlled by the period set of the word [8]. Hence, the need for enumerating Γ_n appears in diverse domains of the literature [21, 22].

The question of enumerating Γ_n is non trivial since Γ_n grows exponentially, as shown in [8], which provided the first upper and lower bound on the logarithm of its cardinality, which is denoted κ_n. The sequence of integers formed by κ_n in function of string length n has an entry in the OEIS. Even the most recent asymptotic upper and lower bounds of $\log(\kappa_n)/\log_2(n)$ are not close to known values of this ratio. At least, the convergence of this ratio, which was conjectured in 1981, was recently proven in 2023 [25]. Currently, only a dynamic programming algorithm exists to enumerate Γ_n, but it suffers from high space complexity [24].

In this work, we propose an incremental approach that computes Γ_n from Γ_{n-1} and uses linear space in n. Our approach needs a *certification* function, which can tell if a subset of

$\{0, 1, \ldots, n-1\}$ is a valid period set or not. Three incremental algorithms that differ by their certification function are presented. They allows one to compute Γ_n for some values considered in real world applications, and thus to investigate combinatorial and statistical properties of Γ_n and of its cardinality.

Plan. In Section 2, we introduce a notation, preliminary results, and review some known results. In section 3, we present the general framework of the incremental algorithm for computing Γ_n, and two variants of it. In section 4 an algorithm for binary realization of period set is explained; it can also be used in the incremental algorithm. In section 5, notions of fate of a period set are defined. Finally, in section 6, we show visualization of Γ_n as a lattice to illustrate these notions and plots interesting parameters related to Γ_n, before concluding with open questions.

2 Related works, notation and preliminary results.

2.1 Notation

Here we introduce a notation and basic definitions.

For two integers $p, q \in \mathbb{N}_{>0}$, we denote the fact that p divides q by $p \mid q$ and the opposite by $p \nmid q$. We consider that strings and arrays are indexed from 0. We use $=$ to denote equality, and $:=$ to denote a definition.

An *alphabet* Σ is a finite set of *letters*. A finite sequence of elements of Σ is called a *word* or a *string*. The set of all words over Σ is denoted by $\Sigma^\star$, and ε denotes the empty word (the only word on length 0). For a word x, $|x|$ denotes the *length* of x. Let n be an integer. The set of all words of length n is denoted by Σ^n. Given two words x and y, we denote by $x.y$ the *concatenation* of x and y. For a word w, we define the powers of w inductively: $w^0 := \epsilon$ and for any $n > 0$, $w^n := w.w^{n-1}$. A word u is *primitive* if there exists no word v such that $u = v^k$ with $k \geq 2$.

Let $u := u[0\,..\,n-1] \in \Sigma^n$. For any $0 \leq i \leq j \leq n-1$, we denote by $u[i-1]$ the i^{th} symbol in u, and by $u[i\,..\,j]$, the substring starting at position i and ending at position j. In particular, $u[0\,..\,j]$ denotes a prefix and $u[i\,..\,n-1]$ a suffix of u.

Let $x, y \in \Sigma^*$ and let j be an integer such that $0 \leq j \leq |x|$ and $j \leq |y|$. If $x[n-j\,..\,|x|-1] = y[0\,..\,j-1]$, then the *merge* of x and y with offset j, which is denoted by $x \oplus_j y$, is defined as the concatenation of $x[0\,..\,n-j-1]$ with y. I.e., $x \oplus_j y := x[0\,..\,n-j-1]y$.

2.1.1 Periodicity

In this subsection, we define the concepts of period, period set, basic period, and autocorrelation, and then review some useful results.

▶ **Definition 1** (Period/border). *The string $u = u[0\,..\,n-1]$ has period $p \in \{0, 1, \ldots, n-1\}$ if and only if $u[0\,..\,n-p-1] = u[p\,..\,n-1]$, i.e. for all $0 \leq i \leq n-p-1$, we have $u[i] = u[i+p]$. Moreover, we consider that $p = 0$ is a period of any string of length n. The substring $u[p\,..\,n-1]$ is called a* border.

Zero is also called the trivial period. The *period set* of a string u is the set of all its periods and is denoted by $P(u)$. The *weight* of a period set is its cardinality. The smallest non-zero period of u is called its *basic period*. When $P(u) = \{0\}$, we consider that its basic period is the string length n.

Let $\Gamma_n := \{Q \subset \{0, 1, \ldots, n-1\} \mid \exists u \in \Sigma^n : Q = P(u)\}$ be the set of all period sets of strings of length n. We denote its cardinality by κ_n. The period sets in Γ_n can be partitioned according to their basic period; thus, for $0 \leq p < n$, we denote by $\Gamma_{n,p}$ the subset of period sets whose basic period is p, and by $\kappa_{n,p}$ the cardinality of this subset.

For the most important properties on periods, we refer the reader to [8, 15] and Appendix F.1. Below we recall the characterization of period sets from [8], and state the version for strings of the famous Fine and Wilf (FW) theorem [6]. We refer the reader to [24, 25] for more properties on period sets.

2.2 Related works

The notion of overlap in strings is crucial in many areas and applications, among others: combinatorics, bioinformatics, code design, or string algorithms.

In cryptography and network communication, the so-called prefix-free, bifix-free, and cross-bifix-free codes are used for synchronization purposes. Their design require to select a set of words that are mutually non overlapping (aka unbordered). This topic has been studied for long: the seminal construction algorithm from Nielsen was published in 1973 [18] together with a note on the expected duration of a search for a fixed pattern in random data [19], thereby linking explicitly pattern matching and code design. Improved algorithms for such code design were published until recently e.g. [2, 1].

Many aspects of periodicity of words were and are still extensively studied in combinatorics giving rise to a huge literature [5, 7]. For instance, the periods of random words were investigated in [11] and the Fine and Wilf (FW) theorem was generalized to more than two words [4]. Some literature has been devoted to constructing extremal FW-words relative to a subset of periods: Given a set of integers R, the question is find the longest word w such that the period set of w includes R, but does not include the gcd of periods in R. Algorithms were proposed and gradually improved in [29, 30] among others. This question is related to our question regarding the fate of period set when the string length n increases.

In bioinformatics, string overlaps are central in the question of DNA or genome assembly. When a genome is broken into pieces and then sequenced, one gets hundreds of millions of reads, which are strings over a 4-letter alphabet. One then needs to compute overlaps between reads, represent these overlaps in graph and search for a Hamiltonian or Eulerian path (depending on the graph) satisfying some length or overlap conditions to infer the full sequence of the genome; see [9, 16] for more pointers. The comparison of sequences and the statistical issues regarding inference of motifs, which both look at the set of k-mers occurring in sequences, are also related to periodicity [27, 26, 31].

Now let us review the most closely related literature to our question. In a seminal work [8], Guibas and Odlyzko defined the notion of autocorrelation of u, which encodes the period set in a binary vector of length n, where `1` at position i indicates that i is a period of u. The binary encoding gives the length n, but is otherwise equivalent to the notion of set period [1]. They have exhibited a recursive characterization of an autocorrelation, which runs in linear time in n. They have provided lower and upper bounds for $\log(\kappa_n)/\log_2(n)$, and conjectured that their lower bound was also an upper bound. They also proposed an algorithm to compute the number of strings in Σ^n that share the same period set, which they termed the *population* of a period set. A key result of their work is the *alphabet independence* of Γ_n: Any alphabet of size at least two gives rise to the same set of period sets, i.e., to Γ_n. Of course, if the alphabet is a singleton all questions mentioned here become trivial. In the

[1] We will see in 5 that a period set can belong to several Γ_n for several values of n.

sequel, let us consider that $\mathrm{card}(\Sigma) > 1$. Note that their characterization of period sets was re-discovered a few years later [13].

Circa 20 years later, Halava et al. gave a simpler proof of the alphabet independence result [10] by solving the following question. For any period set Q of Γ_n, let v be a word over an alphabet of cardinality larger or equal to 2 such that $P(v) = Q$; then compute a binary word u such that $P(u) = Q$. Indeed, they exhibited a linear time algorithm that computes such a word u from v.

At the same period, other authors investigated the structure of Γ_n to show that it is a lattice that does not satisfy the Jordan-Dedekind condition [24]. Moreover, they designed an enumeration algorithm for Γ_n that uses a dynamic programming approach. Given that κ_n is exponential in n, their enumeration algorithm also is, but its main drawback lies in memory usage, which requires to store all $\Gamma(i)$ for $0 < i \leq \lfloor 2n/3 \rfloor$. With combinatorial arguments about the number of binary partitions of an integer, they provided improved the lower bounds for for $\log(\kappa_n)/\log_2(n)$. Many concepts and results have been extended to words with don't care symbols, e.g. [3]. In 2023, the conjecture stated by Guibas and Odlyzko regarding this ratio was finally proven to be correct, thereby implying that for $\log(\kappa_n)/\log_2(n)$ converges towards $1/(2\log(2))$ when n tends to infinity [25].

2.3 Rule based characterization

In their seminal paper, Guibas and Odlyzko characterized autocorrelations by three conditions: they must

1. start with a 1 (i.e., zero is a period)
2. satisfy the Forward Propagation Rules (FPR)
3. satisfy the Backward Propagation Rule (BPR)

Let us formulate the FPR and BPR in terms of sets (rather than in term of binary vector as in [8]). Let P a subset of $\{0, 1, \ldots, n-1\}$.

▶ **Definition 2.** *P satisfies the FPR iff for all pairs p, q in P satisfying $0 \leq p < q < n$, it follows that $p + i(q-p) \in P$ for all $i = 2, \ldots, \lfloor (n-p)/(q-p) \rfloor$.*

▶ **Definition 3.** *P satisfies the BPR iff for all pairs p, q in P satisfying $0 \leq p < q < 2p$, and $(2p-q) \notin P$, it follows that $p - i(q-p) \notin P$ for all $i = 2, \ldots, \min(\lfloor p/(q-p) \rfloor, \lfloor (n-p)/(q-p) \rfloor)$.*

In [8], the authors give a version for strings of the famous Fine and Wilf theorem [6], a.k.a. the periodicity lemma. A nice proof was provided by Halava and colleagues [10].

▶ **Theorem 4** (Fine and Wilf)**.** *Let p, q be periods of $u \in \Sigma^n$. If $n \geq p + q - \gcd(p, q)$, then $\gcd(p, q)$ is a period of u.*

We can reformulate this theorem as a condition that must be satisfied by a period set P of Γ_n.

▶ **Theorem 5.** *Let any pair p, q of periods of P such that $\gcd(p, q) \notin P$, and define $FW(p, q) := p + q - \gcd(p, q)$. Then $FW(p, q)$ must be strictly larger than n.*

We call $FW(p, q)$ the *Fine and Wilf (FW) limit* of (p, q), and the fact that $FW(p, q)$ must be larger than n, the *FW condition.*

We define the notion of *nested set.* It helps formulating definitions and properties that were originally expressed as suffixes of autocorrelations in [8].

▶ **Definition 6.** *Let $n > 0$ and P be a subset of $\{0, 1, \ldots, n-1\}$. Let q be an element of P. Let us denote the nested set of P starting at period q as P_q:*

$$P_q := \{(r - q) \textit{ for each } r \in P \textit{ such that } r \geq q\}.$$

By construction P_q starts with 0; moreover, if we choose $q = 0$ then $P_q = P$. Now assume that P is valid PS of length n, and q a period of P. Then, by Theorem 9 (which we recall in Section 3), we have that P_q is a valid PS of length $(n - q)$.

2.4 Checking the FPR and the BPR

Let $n > 0$ and P be a subset of $\{0, 1, \ldots, n-1\}$. We assume that P is given as an ordered array. The complexity for checking the FPR or the BPR for P, has, to our knowledge not been previously addressed. For any pair $p < q$, we call their difference $(q - p)$, an offset.

Checking the BPR. Here, we demonstrate a property that relates the BPR to the FW theorem. Precisely, if BPR is violated for some pair (p, q) at length n, with period $r := p - i(q - p)$ for some i, then the pair $(p - r, q - r)$ violates the FW condition of Theorem 5 in the nested set of length $(n - r)$.

▶ **Lemma 7.** *Let (p, q) be a pair of integers that violate the BPR, and let $i \geq 2$ such that $r := p - i(q - p) \in P$. Then the pair $(p - r, q - r)$ violates the FW condition for length $(n - r)$.*

Proof. Let $P \in \Gamma_n$. Let p, q in P satisfying $0 \leq p < q < 2p$ be such that $(2p - q) \notin P$. Assume (p, q) violates the BPR. Then, there exists i in $[2, \ldots, \min(\lfloor p/(q-p), \rfloor, \lfloor (n-p)/(q-p) \rfloor)]$, such that $p - i(q - p) \notin P$. If several such integers exist, choose i as their minimum, and define $r := p - i(q - p)$. We will show that the nested period set of P for length $(n - r)$ is not valid since two of its periods violates the FW condition, which would require their gcd as an additional period, thereby implying that P is not a valid period set for length n, a contradiction. Since i is chosen minimal, we have that $\gcd(p, q)$ is not in P by the definition of the BPR. Note that $p - r = i(q - p)$ and $q - r = (i + 1)(q - p)$. Thus, one gets $gcd(p - r, q - r) = q - p$, and the FW limit of $(p - r, q - r)$ equals $2i(q - p)$. Indeed, $FW(p - r, q - r) := p - r + q - r - gcd(p - r, q - r) = 2p - 2r = 2i(q - p)$. By hypothesis, we have:

$$\begin{array}{rlcl} & i & \leq & (n-p)/(q-p) \\ \Leftrightarrow & p + i(q-p) & \leq & n \\ \Leftrightarrow & r + 2i(q-p) & \leq & n \\ \Leftrightarrow & FW(p-r, q-r) & \leq & n - r \end{array}$$

meaning that $(p - r, q - r)$ violates the FW condition of Theorem 5 for length $(n - r)$. ◀

In algorithmic terms, checking the BPR can be done by checking the FW condition of Theorem 5 in each nested set.

Checking the FPR. Some properties are explained in Appendix D and lead to this Lemma.

▶ **Lemma 8.** *Let P is a subset of $[0, \ldots, n-1]$, that is ordered, and has zero as first period. Checking the FPR for P in general takes at most $O(n \log_2(n))$time.*

3 Incremental enumeration framework

3.1 Rationale for an incremental algorithm to enumerate Γ_n

The rule based characterization of autocorrelations from [8] implies a *special substring* property. Indeed, Lemma 3.1 in [8] states: If a binary vector v satisfies the forward and backward propagation rules, then so does any prefix or suffix of v. As the characterization also requires that an autocorrelation has its first bit equal to one, or equivalently that zero belongs to any period set, one gets the following theorem.

▶ **Theorem 9.** *Let v be an autocorrelation of length n. Any substring $v_i \dots v_j$ of v with $0 \leq i \leq j < n$ such that $v_i = 1$ is an autocorrelation of length $j - i + 1$.*

Applying Theorem 9 to a prefix of v, one gets for any $n > 0$: The prefix of length $(n-1)$ from an autocorrelation of length n is an autocorrelation of length $(n-1)$. In terms of period sets, this statement can be reformulated as:

▶ **Corollary 10.** *If P is a period set of Γ_n, then $P \setminus \{n-1\}$ belongs to Γ_{n-1}.*

First, this means that, knowing Γ_n, it is easy to compute Γ_{n-1}. It suffices to consider each element of Γ_n in turn (or in parallel) and to eventually remove the period $(n-1)$ from it (i.e., if $(n-1)$ belongs to it) to obtain an element of Γ_{n-1}. With this procedure one can obtain the same element of Γ_{n-1} twice, and one must keep track of this to avoid redundancy. Conversely, we get the Lemma that underlies the incremental approach for computing Γ_n:

▶ **Lemma 11.** *Let P be a period set of Γ_{n-1}. Then, a period set Q of Γ_n can only be of two alternative forms: either P or $P \cup \{n-1\}$.*

3.2 Incremental algorithm framework

Lemma 11 suggests an approach for computing Γ_n using Γ_{n-1}. Consider each P from Γ_{n-1}, and certify that the candidate sets, P and $P \cup \{n-1\}$, are valid period sets of Γ_n. Clearly, Algorithm 1 presents the general incremental algorithm for Γ_n, where `certify` denotes the certification function used. This function must take as input n and a subset of $\{0, 1, \dots, n-1\}$, and check the validity of this subset as a period set for length n.

Algorithm 1 IncrementalGamma

```
Input : n > 1: integer; Γ_{n−1}: the set of period sets for length n − 1
Output: Γ_n: the set of period sets for length n;
G := ∅;                                      // G: variable to store Γ_n
for all P ∈ Γ(n − 1) do
    if certify( P, n ) then // check that P is valid at length n
        insert P in G;
    Q := P ∪ {n − 1} // build extension P with period n − 1;
    if certify( Q, n ) then // check that Q is valid at length n
        insert Q in G;
return G;
```

The recursive predicate Ξ from [8] (cf. Appendix F.2) is one possible certification function, which does exactly what is required for any subset of $\{0, 1, \dots, n-1\}$, in linear time [8]. This means that Algorithm 1 correctly computes Γ_n from Γ_{n-1}. However, since we know that P belongs to Γ_{n-1}, the candidate sets are not **any subset** of $\{0, 1, \dots, n-1\}$, but specific ones that already satisfy some constraints for length $(n-1)$. Therefore, finding alternative certification functions is interesting.

Besides its simplicity, the main advantage of Algorithm 1, compared to the dynamic programming enumeration algorithm of [23], is its space complexity. Here, the computation considers each period set P from Γ_{n-1} in turn (and independently from the others), executes twice the certification function for P and Q; this implies that the memory required, besides storage of P, Q, is the one used by the certification function. With the predicate Ξ, it is linear in n, so $O(n)$ space. The time complexity is proportional to κ_n (i.e., the cardinality of Γ_n) times the running time of the certification function, which yields the following theorem.

▶ **Theorem 12.** *One has*

1. *Algorithm 1 using any correct certification correctly computes Γ_n from Γ_{n-1}.*
2. *Using the predicate Ξ as certification function, it runs in $O(n\kappa_n)$ time and $O(n)$ space.*

Moreover, it is worth noticing that Algorithm 1 is embarrassingly parallelizable.

3.3 Alternative certification function.

Let us propose a **second certification function** that derives from the rule based characterization of autocorrelations also presented in [8]. It states that a period set of Γ_n must i/ contains the trivial period zero, ii/ satisfy the Forward Propagation Rule (FPR), and iii/ the Backward Propagation Rule (BPR). The rule based characterization is shown to be equivalent to the predicate Ξ in Theorem 5.1 from [8] (see Appendix F.2).

The pseudo-code of the incremental algorithm for computing Γ_n using the rule based certification function is shown and explained in Algorithm 3 in Appendix B.

4 Constructive certification of a period set

Let R be subset of $\{0, 1, \ldots, n-1\}$. We say that a word u *realizes* R if $P(u) = R$. Another interesting certification function is: to attempt to build a word u that realizes R; if the attempt succeeds, R is a valid period set. Given the alphabet independence of Γ_n, we restrict the search to binary strings.

Below we present an algorithm for the *binary realization* of a set (see Algorithm 2). Using it as a certification function in Algorithm 1, the latter will compute Γ_n from Γ_{n-1} and also yield one realizing string for each period set.

4.1 Binary realization of a subset of $\{0, 1, \ldots, n-1\}$

Algorithm 2 computes a word u that realizes a set P for length $n > 0$, or returns the empty word ϵ if P is not a valid period set of Γ_n. For legibility, the preliminary checks on P are not written in Algorithm 2: they include checking that P is a subset of $[0, \ldots, n-1]$, is ordered, and has zero as first period. The word u is written over the alphabet $\{\mathtt{a}, \mathtt{b}\}$.

The algorithm considers elements of P backwards, starting with largest integer first, since P is ordered. At each execution of the for loop, it considers the current integer $P[i]$ as a period and builds a suffix of u of length $n - P[i]$ (variable *lg*). In fact, it considers a potential larger and larger nested sets, and computes a suffix of u for this length. At the end of the for loop, the variable *suffix* contains a string of length *lg* realizing the nested set. Note that algorithm uses three variables (whose names start with *prev*) to store the length, the inner period, and the suffix obtained with the previous period.

Algorithm 2 Binary Realization

Input: $n > 0$: integer; P: a subset of $[0, 1, \ldots, n-1]$ including 0, in a sorted array
Output: a **binary** string realizing P at length n xor the empty string otherwise;

1 $k := \mathrm{card}(P)$; `// k: cardinality of P`
2 **if** $k = 1$ **then return** $\mathtt{a.b}^{(\mathtt{n-1})}$ `// trivial case where` $P = \{0\}$;
`// processing the largest period and init. variables`
3 $\mathrm{prevLg} := n - P[k-1]$; $\mathrm{prevIP} := \mathrm{prevLg}$; $\mathrm{prevSuffix} := \mathtt{a.b}^{(\mathrm{prevLg}-1)}$;
4 **for** *i going from* $k-2$ *to* 0 **do**
5 $\quad \mathrm{lg} := n - P[i]$;
6 $\quad \mathrm{innerPeriod} := P[i+1] - P[i]$;
7 $\quad$ **if** *innerPeriod* $<$ *prevIP* **then return** ϵ ;
8 $\quad$ **if** $lg \leq 2\times$ *prevLg* **then** `// condition for case 1`
9 $\quad\quad$ **if** (*innerPeriod* $=$ *prevIP*) *OR* ((*prevIP* $\nmid$ *innerPeriod*) *AND* ((*innerPeriod* $=$ *prevLg*) *OR* (*prevSuffix has period innerPeriod*))) **then**
$\quad\quad\quad$ `// suffix := a prefix of prevSuffix concat. with prevSuffix`
10 $\quad\quad\quad \mathrm{suffix} := \mathrm{prevSuffix}[0..\mathrm{innerPeriod}-1]\ .\ \mathrm{prevSuffix}$;
11 $\quad\quad$ **else return** ϵ `// invalid case for length lg;`
12 $\quad$ **else** `// condition for case 2`
$\quad\quad$ `// suffix := prevSuffix newsymbols prevSuffix`
13 $\quad\quad \mathrm{nb} := \mathrm{lg} - 2\times \mathrm{prevLg}$;
14 $\quad\quad \mathrm{newPrefix} := \mathrm{prevSuffix}\ .\mathtt{a}^{\mathtt{nb}}$;
15 $\quad\quad$ **if** *newPrefix is not primitive* **then**
16 $\quad\quad\quad \mathrm{newPrefix} := \mathrm{prevSuffix}\ .\mathtt{a}^{(\mathtt{nb}-1)}\mathtt{b}$;
$\quad\quad$ `// Invariant: newPrefix is primitive`
17 $\quad\quad \mathrm{suffix} := \mathrm{newPrefix}\ .\ \mathrm{prevSuffix}$;
$\quad$ `// update variables`
18 $\quad \mathrm{prevLg} := \mathrm{lg}$; $\mathrm{prevIP} := \mathrm{innerPeriod}$; $\mathrm{prevSuffix} := \mathrm{suffix}$;
19 **return** *suffix*;

The base case is processed before the loop and consider the nested set for $P[k-1] = max(P)$ for the length $n - max(P)$ without any period. Hence, the suffix $\mathtt{a.b}^{(\mathrm{prevLg}\ \text{-}1)}$ is a realization for nested set $\{0\}$ for length $n - max(P)$.

In the for loop the key variable is the *innerPeriod*, which equals the offset $P[i+1] - P[i]$, which is the basic period of the current nested set. If innerPeriod $<$ prevIP then the FPR is violated and the algorithm returns ϵ (line 7). Two cases are considered depending on whether the current length is smaller twice the previous length (case 1) or not (case 2). Because of the notion of period, the suffix must start and end with a copy of *prevSuffix*. The construction of the suffix depends on the case. In case 1, the two copies of *prevSuffix* are concatenated or overlap themselves, and some additional conditions are required (line 9). These conditions are dictated by the characterization of period set from [8] (see the predicate Ξ in Appendix F.2). Whenever one is not satisfied, Algorithm 2 returns the empty word as expected. In case 2, the two copies of *prevSuffix* must be separated by *nb* additional symbols (to be determined). One builds a *newPrefix* that starts with *prevSuffix* followed by $\mathtt{a}^{nb}$, and one checks whether *newPrefix* is primitive. This *newPrefix* is the part that ensures the suffix will have *innerPeriod* as a period. The primitivity is required, since *newPrefix* may have a proper period, but this period shall not divide *innerPeriod*. If primitivity is not satisfied, then changing the last symbol of *newPrefix* by a `b` will make it primitive. This is enforced by Lemma 3 from [10], which states that for any binary word w, w`a` or w`b` is primitive. So we know that at least one of the two forms of *newPrefix* is primitive as necessary. It can be that both are primitive and suitable. Finally, we build the current *suffix* by concatenating *newPrefix* with *prevSuffix*.

Complexity First, in case 1, checking the condition "*prevSuffix has period innerPeriod*" can be done in linear time in $|prevSuffix|$ (which is $\leq n$). Overall, this can be executed $card(P)$ times. Second, the primitivity test performed in case 2 takes a time proportional to the length of the string *newPrefix*. However, the sum of these lengths, for all iterations of the loop, is bounded by n. Other instructions of the for loop take constant time. Overall, the time complexity of Algorithm 2 by $O(card(P) \times n)$. However, when Algorithm 2 is plugged in Algorithm 1 it processes special instances: either P or $Q := P \cup \{n-1\}$, with $P \in \Gamma_{n-1}$. Then, the time taken by all verifications of condition "*prevSuffix has period innerPeriod*" for all cases 1 is bounded by n, due to the properties of periods that generate more than two repetitions in a string (see Lemma 15 and Lemma 2 from [10]). For the instances processed in Algorithm 1, the time complexity of Algorithm 2 is $O(card(P) + n)$ or $O(n)$.

Remark It is possible to modify Algorithm 2 to build, instead of a binary word, a realizing string that maximizes the number of distinct symbols used in it. Indeed, new symbols are used only in the base case and in case 2. Each time, it is possible to choose symbols that have not been used earlier in the algorithm, and thus to maximize the overall number. Note that this would remove the need of the primitivity test in case 2.

4.2 Examples of binary realization

We take the case of a valid period set at length $n = 9$ that becomes invalid at $n = 10$, and show the traces of execution for both lengths. Let $P = \{0, 3, 6, 8\}$ and first $n = 9$. The operator $\oplus_j$ merges the two strings with an offset of length j if the corresponding prefix and suffix are equal, for any appropriate integer j. So when $n = 9$, the merge $v = w \oplus_3 w$ with $w = abaaba$ is feasible since w has period 3. When $n = 10$, the merge $w = y \oplus_3 y$ with $y = abab$ is not possible since $a \neq b$. The trace for $n = 10$ is in Appendix C.

period	length	inner period	case	suffix	valid
8	9-8 = 1	9-8 = 1	2	$z = a$	true
6	9-6 = 3	8-6 = 2	2	$y = zbz = aba$	true
3	9-3 = 6	6-3 = 3	2	$w = yy = abaaba$	true
0	9-0 = 9	3-0 = 3	1	$v = w \oplus_3 w = (aba)^3$	true

5 Fate and dynamics of period sets

One interest of incremental algorithms is to shed light on the dynamics of the Γ_n when n increases, both in terms of new and dying period sets, as well as on the structure of Γ_n. As mentioned in introduction, Γ_n is a lattice under inclusion; the union and the intersection of two period sets are period sets [24]. Even if the cardinality of Γ_n increases with n, the growth is not regular. It is worth investigating the local dynamics of Γ_n when n changes, and for this we define the fate of period sets.

5.1 Fate of a period set when n increases

The incremental algorithms presented above show that Γ_{n-1} and Γ_n share some period sets, and that other period sets are derived by an extension, that is are of the form $P \cup \{n-1\}$.

Let P be a period set of Γ_{n-1}. The maximal period in P, denoted $max(P)$, determines the first length at which P exists: indeed, the *birth* of P occurs in $\Gamma_{max(P)+1}$. When the length increases, say from $n-1$ to n, what can be the fate of P? Only three possibilities exist:

1. either P remains valid at length n,
2. or P has an *extension* with period $n-1$ at length n,
3. or P *dies* (i.e., is neither case 1 nor case 2), see Definition 14 in Appendix B.

Note that cases 1 and 2 are not exclusive from each other. Let us illustrate these with the following example.

▶ **Example 13.** For instance, $\{0, 3, 6\}$ is born in Γ_7 and also belongs to Γ_8 and Γ_9; its extension $\{0, 3, 6, 7\}$ also belongs to Γ_8. This extension is not compulsory since $\{0, 3, 6\}$ also belongs to Γ_8. From a dynamic view point, one can say that $\{0, 3, 6\}$ from Γ_7 *generates* both $\{0, 3, 6\}$ and $\{0, 3, 6, 7\}$ in Γ_8. On the contrary, $\{0, 4, 6\}$ belongs Γ_7, but dies at $n = 8$, since the pair of periods $(4, 6)$ satisfies the FW condition at that length and would require to add $gcd(4, 6)$ as a new basic period. Last, $\{0, 2, 4, 6\}$ belongs to Γ_7, Γ_8, and generates $\{0, 2, 4, 6, 8\}$ in Γ_9 because the extension with period 8 is required by the FPR.

The fate of depends on

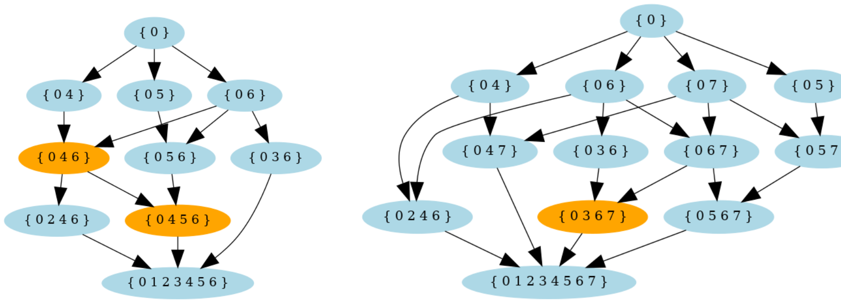
1. the smallest length at which an extension will be required (i.e., as a consequence of the FPR a new period is added to P at some length); we call it the *extension limit* of P,
2. the smallest length at which some pairs of periods violates the BPR (we call it the *Recursive FW limit*).

These two lengths depend only on the period set, and can thus be computed as soon as P is born. We provide in Appendix E two algorithms to compute these limits.

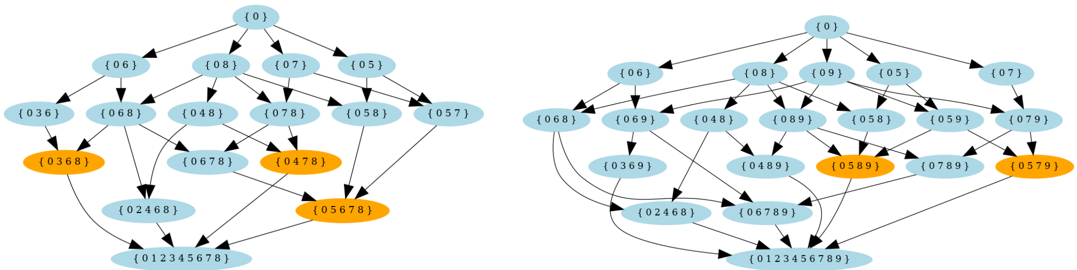
Figure 1 shows for $n = 7, \ldots, 11$, the set Γ_n represented as a lattice of period sets with the inclusion relationship. The out-going arrows of a period set point to its successors in the lattice. The dying period sets of Γ_n, that is those that do not exist at length $n+1$, are shown in orange background. The number of dying period sets is not monotonically increasing: it equals 2 in Γ_7, 1 in Γ_8, 3 in Γ_9, 2 in Γ_{10}, and 8 in Γ_{11}, for instance. Clearly, it does not prevent the cardinality of Γ_n to increase monotonically. An interesting question for future work is to find the function that for n gives the number of dying period sets of Γ_n, and how these are distributed with respect to their basic period.

6 Conclusion and exploration of Γ_n: distribution of period sets with respect to basic period and weight

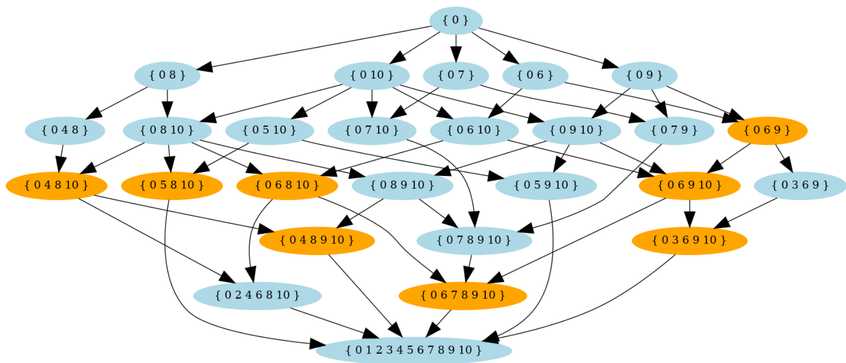
The key element of a period set is its basic period, which defines the first level of periodicity in a word. How period sets in Γ_n are distributed according to their basic period is non trivial. Enumerating Γ_n allows inspecting this distribution. The left plot in Figure 2 displays $\kappa_{n,p}$, the counts of period sets for all possible basic periods p, in Γ_{60}. In predicate Ξ [8], as well as in the dynamic programming algorithm that enumerates Γ_n [24], one separates period sets depending on the basic period being larger than $\lfloor n/2 \rfloor$ (case **b**) or smaller than or equal to it (case **a**). The smooth decrease of counts beyond the basic period equals to half of the string length is explained by the combinatorial property that links number of period sets in case **b** and the number of binary partitions of an integer (see Lemma 5.8 in [24]). However,



(a) Γ_7 and Γ_8



(b) Γ_9 and Γ_{10}



(c) Γ_{11}

Figure 1 Lattices representations of Γ_7 and Γ_8 **(a)**, Γ_9 and Γ_{10} **(b)**, and of Γ_{11} **(c)**. Each node contains a period set (a list of periods separated by spaces). Those whose last period equals $(n-1)$ are obtained by extension of period set from Γ_{n-1}, and those nodes in orange background are dying period sets at length $n+1$.

the distribution of counts for all period sets in case **a**, still requires some investigation and statistical modeling. Here, we observe that between basic period 1 and 30, $\kappa_{n,p}$ increases globally with the basic period p, but locally $\kappa_{n,p}$ increases and then decreases to reach local maxima when p divides the string length n (e.g. see the peaks at $p = 10, 12, 15, 20, 30$, which correspond to period sets of case **a**).

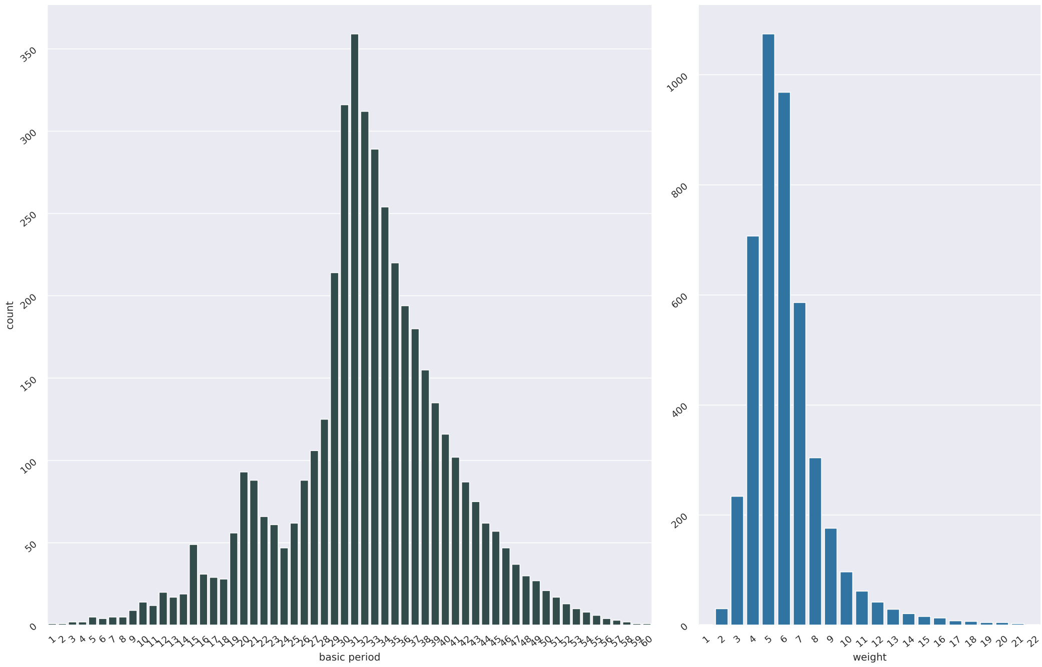


Figure 2 Distribution in Γ_{60} of the number of period sets by basic period (left) and by weight (right), for string length of $n := 60$. Beyond basic period 30, the counts decrease smoothly with the basic period. Between basic period 1 and 30 the counts increase to a local maximum when the basic period reaches $\lfloor n/x \rfloor$ for $1 < x \leq 12 =$ (e.g. basic periods 10, 12, 15, 20, 30). The distribution by weight (right) is limited to weight below 22; it is unimodal and right skewed towards low weights.

Other works have investigated combinatorial parameters that control the number of periods of a word [7]. Thanks to enumeration of Γ_n one can study the real distribution of weight of period sets and how it evolves with n. The right plot of Figure 2 displays the number of period sets having the same weight (i.e. same number of periods) for $n = 60$. This distribution is right skewed and illustrates the constraints imposed by multiple periods. Similar figures for other string lengths are shown in Appendix A.

Conclusion. We provide algorithms to enumerate Γ_n incrementally with low space requirement, and an algorithm for binary realization of a period set. They allow to inspect Γ_n and to visualize how parameters like the weight or the basic period impact the number of PS. We define the fate of a PS and propose to study the dynamics of Γ_n when n increases. Many questions remain: how can the recursive FW and extension limits of PS of Γ_{n-1} be used to speed up the incremental enumeration of Γ_n? Can we exploit binary realizing strings to ease enumeration or to unravel how population sizes evolve with n? Among directions for future work, finding algorithms to enumerate PS for generalizations of words, like partial words or multidimensional words (aka matrices) is interesting. As seen in Figure 1, the number of PS that die in function of n is not monotonically increasing; thus understanding the sequences of κ_n and $\kappa_{n,p}$ is both stimulating and challenging (see also Figures 2, 3-5).

References

1. Dragana Bajic and Tatjana Loncar-Turukalo. A simple suboptimal construction of cross-bifix-free codes. *Cryptography and Communications*, 6(6):27–37, 2014. doi:10.1007/s12095-013-0088-8.
2. Stefano Bilotta, Elisa Pergola, and Renzo Pinzani. A new approach to cross-bifix-free sets. *IEEE Transactions on Information Theory*, 58(6):4058–4063, 2012. doi:10.1109/TIT.2012.2189479.
3. Francine Blanchet-Sadri, Justin Fowler, Joshua D. Gafni, and Kevin H. Wilson. Combinatorics on partial word correlations. *J. Comb. Theory, Ser. A*, 117(6):607–624, 2010. doi:10.1016/j.jcta.2010.03.001.
4. M.Gabriella Castelli, Filippo Mignosi, and Antonio Restivo. Fine and wilf's theorem for three periods and a generalization of sturmian words. *Theoretical Computer Science*, 218(1):83–94, April 1999. URL: http://dx.doi.org/10.1016/S0304-3975(98)00251-5, doi:10.1016/s0304-3975(98)00251-5.
5. Andrzej Ehrenfeucht and D.M. Silberger. Periodicity and unbordered segments of words. *Discrete Mathematics*, 26(2):101–109, 1979. URL: http://dx.doi.org/10.1016/0012-365X(79)90116-X, doi:10.1016/0012-365x(79)90116-x.
6. N. J. Fine and H. S. Wilf. Uniqueness theorems for periodic functions. *Proc. Amer. Math. Soc.*, 16:109–114, 1965.
7. Daniel Gabric, Narad Rampersad, and Jeffrey Shallit. An inequality for the number of periods in a word. *International Journal of Foundations of Computer Science*, 32(05):597–614, Jun 2021. doi:10.1142/s0129054121410094.
8. Leo J. Guibas and Andrew M. Odlyzko. Periods in strings. *J. of Combinatorial Theory series A*, 30:19–42, 1981. doi:10.1016/0097-3165(81)90038-8.
9. Dan Gusfield. *Algorithms on Strings, Trees and Sequences*. Cambridge University Press, 1997.
10. Vesa Halava, Tero Harju, and Lucian Ilie. Periods and binary words. *J. Comb. Theory, Ser. A*, 89(2):298–303, 2000. URL: https://doi.org/10.1006/jcta.1999.3014, doi:10.1006/JCTA.1999.3014.
11. Stepan Holub and Jeffrey O. Shallit. Periods and borders of random words. In Nicolas Ollinger and Heribert Vollmer, editors, *33rd Symposium on Theoretical Aspects of Computer Science, STACS 2016, February 17-20, 2016, Orléans, France*, volume 47 of *LIPIcs*, pages 44:1–44:10. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPIcs.STACS.2016.44.
12. D.E. Knuth, J.H. Morris, and V.R. Pratt. Fast pattern matching in strings. *SIAM Journal of Computing*, 6:323–350, 1977.
13. Douglas A. Leonard. A prefix problem. *Ars Combinatoria*, 24:51–55, 1987.
14. Paul Leopardi. Testing the Tests: Using Random Number Generators to Improve Empirical Tests. In Pierre L' Ecuyer and Art B. Owen, editors, *Monte Carlo and Quasi-Monte Carlo Methods 2008*, pages 501–512. Springer Berlin Heidelberg, 2009. DOI: 10.1007/978-3-642-04107-5_32. URL: http://link.springer.com/chapter/10.1007/978-3-642-04107-5_32.
15. M. Lothaire, editor. *Combinatorics on Words*. Cambridge University Press, second edition, 1997.
16. Veli Mäkinen, Djamal Belazzougui, Fabio Cunial, and Alexandru I. Tomescu. *Genome-Scale Algorithm Design: Biological Sequence Analysis in the Era of High-Throughput Sequencing*. Cambridge University Press, 2015. doi:10.1017/CBO9781139940023.
17. G. Marsaglia and A. Zaman. Monkey tests for random number generators. *Computers and Mathematics with Applications*, 26(9):1–10, 1993.
18. P. Nielsen. A note on bifix-free sequences (corresp.). *IEEE Transactions on Information Theory*, 19(5):704–706, September 1973. URL: http://dx.doi.org/10.1109/TIT.1973.1055065, doi:10.1109/tit.1973.1055065.
19. P. Nielsen. On the expected duration of a search for a fixed pattern in random data (corresp.). *IEEE Transactions on Information Theory*, 19(5):702–704, September 1973. URL: http://dx.doi.org/10.1109/TIT.1973.1055064, doi:10.1109/tit.1973.1055064.

20 Ora E. Percus and Paula A. Whitlock. Theory and Application of Marsaglia's Monkey Test for Pseudorandom Number Generators. *ACM Transactions on Modeling and Computer Simulation*, 5(2):87–100, April 1995.

21 Sven Rahmann and Eric Rivals. Exact and efficient computation of the expected number of missing and common words in random texts. In *Proc. of CPM 2000*, page 375–387. Springer Berlin Heidelberg, 2000. URL: `http://dx.doi.org/10.1007/3-540-45123-4_31`, `doi:10.1007/3-540-45123-4_31`.

22 Sven Rahmann and Eric Rivals. On the distribution of the number of missing words in random texts. *Combinatorics, Probability and Computing*, 12(01), Jan 2003. URL: `http://dx.doi.org/10.1017/S0963548302005473`, `doi:10.1017/s0963548302005473`.

23 Eric Rivals and Sven Rahmann. Combinatorics of periods in strings. In *Proc. of ICALP 2001*, page 615–626. Springer Berlin Heidelberg, 2001. URL: `http://dx.doi.org/10.1007/3-540-48224-5_51`, `doi:10.1007/3-540-48224-5_51`.

24 Eric Rivals and Sven Rahmann. Combinatorics of periods in strings. *Journal of Combinatorial Theory, Series A*, 104(1):95–113, Oct 2003. URL: `http://dx.doi.org/10.1016/S0097-3165(03)00123-7`, `doi:10.1016/s0097-3165(03)00123-7`.

25 Eric Rivals, Michelle Sweering, and Pengfei Wang. Convergence of the Number of Period Sets in Strings. In Kousha Etessami, Uriel Feige, and Gabriele Puppis, editors, *50th International Colloquium on Automata, Languages, and Programming (ICALP 2023)*, volume 261 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 100:1–100:14, Dagstuhl, Germany, 2023. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. URL: `https://drops.dagstuhl.de/opus/volltexte/2023/18152`, `doi:https://doi.org/10.4230/LIPIcs.ICALP.2023.100`.

26 S. Robin and J.-J. Daudin. Exact distribution of word occurrences in a random sequence of letters. *J. of Applied Probability*, 36:179–193, 1999.

27 Stéphane Robin, François Rodolphe, and Sophie Schbath. *DNA, Words and Models*. Cambridge Univ. Press, 2005.

28 William F. Smyth. *Computating Pattern in Strings*. Pearson - Addison Wesley, 2003.

29 R. Tijdeman and L. Zamboni. Fine and wilf words for any periods. *Indagationes Mathematicae*, 14(1):135–147, March 2003. URL: `http://dx.doi.org/10.1016/S0019-3577(03)90076-0`, `doi:10.1016/s0019-3577(03)90076-0`.

30 R. Tijdeman and L.Q. Zamboni. Fine and wilf words for any periods ii. *Theoretical Computer Science*, 410(30–32):3027–3034, August 2009. URL: `http://dx.doi.org/10.1016/j.tcs.2009.02.004`, `doi:10.1016/j.tcs.2009.02.004`.

31 E. Ukkonen. Approximate string-matching with q-grams and maximal matches. *Theor. Comp. Sci.*, 92(1):191–211, January 1992.

A Exploration of Γ_n: Distribution of the number of period sets by basic period and by weight

Like in Figure 2, we explore how period sets are distributed according to their basic period, and according to their weight for other string lengths. We plot these distributions for $n = 48$, $n = 55$, and $n = 59$ in Figures 3, 4, and 5, respectively. We choose these values because they differ in their number of divisors $48 = 2^4 \times 3$, $55 = 5 \times 11$ and 59 is prime. In essence, both plots for Γ_{48}, Γ_{55}, and Γ_{59} look very similar to those for Γ_{60}. Even for a prime string length, $n = 59$, the distribution of number of period sets in case **a**, shows a maximum at $\lfloor n/2 \rfloor$ and local maxima at $\lfloor n/3 \rfloor$, $\lfloor n/4 \rfloor$ etc.

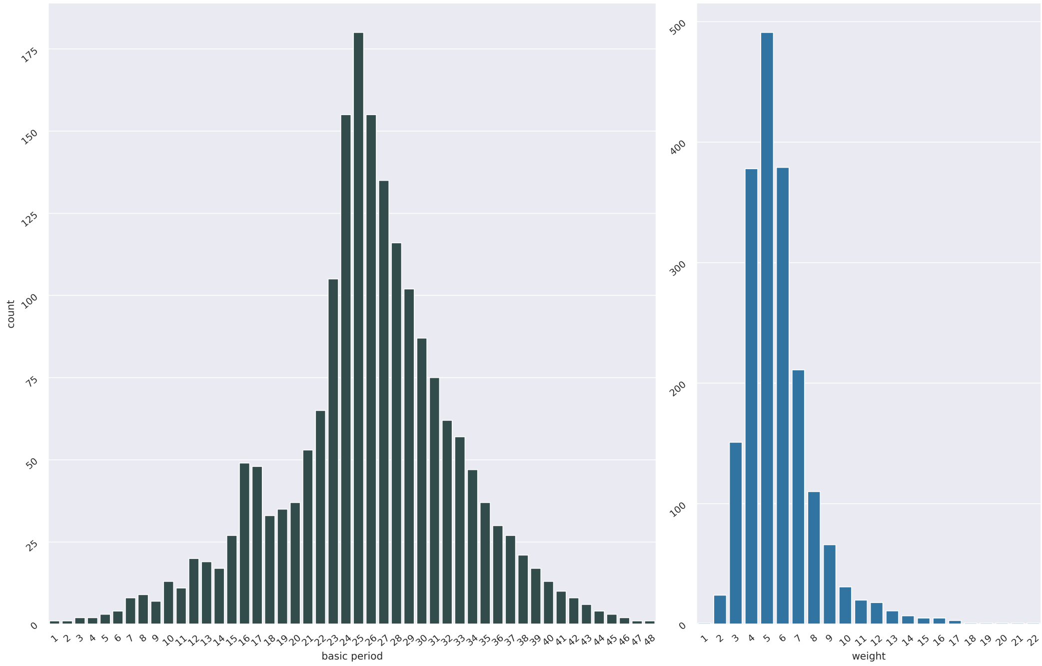


Figure 3 Distribution in Γ_{48} of the number of period sets by basic period (left) and by weight (right), i.e., for string length of $n := 48$.

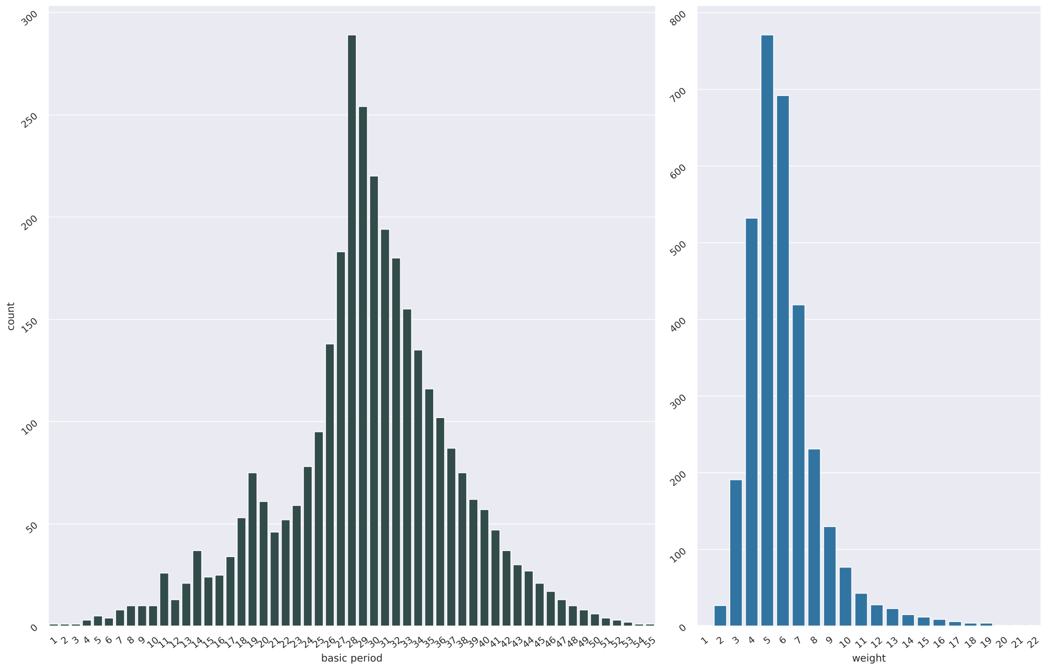


Figure 4 Distribution in Γ_{55} of the number of period sets by basic period (left) and by weight (right), i.e., for string length of $n := 55$.

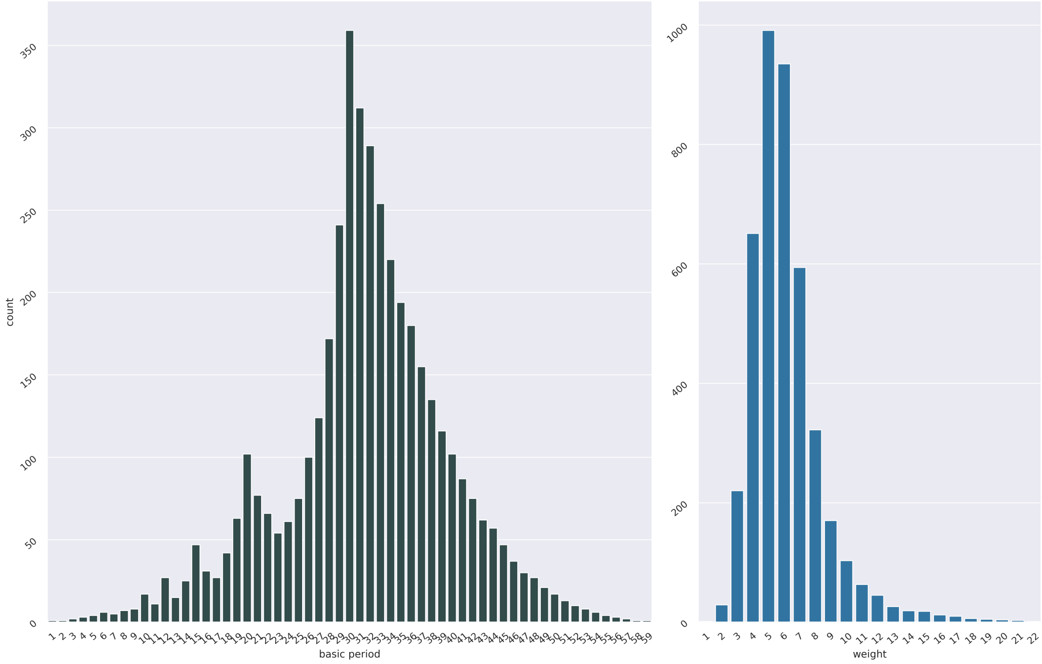


Figure 5 Distribution in Γ_{59} of the number of period sets by basic period (left) and by weight (right), i.e., for string length of $n := 59$.

B Incremental algorithm with rule based certification.

Here, we detail an alternative version of the incremental algorithm, which uses the rule based certification function derived from Theorem 5.1 from [8] (see also below). This is related to subsection 3.3. Algorithm 3 presents the pseudo-code; it uses two functions named *checkFPR* and *checkBPR*, which check if a set of integers satisfies respectively, the Forward and Backward Propagation Rules.

In our case, as the candidate sets include a period set of Γ_{n-1}, they necessarily satisfy the first condition (i). Regarding the FPR, since P belongs to Γ_{n-1}, P satisfies the FPR up to position $n-2$ included; and thus, only period $n-1$ can be required by the FPR. For each possible pair (p, q) considered in the FPR, we only need to check if the FPR formula yields $(n-1)$. Second, for the same reason, when considering the candidate set $P \cup \{n-1\}$, we are sure that the FPR is satisfied.

Let R be any subset of $\{0, 1, \ldots, n-1\}$ containing zero and assuming that R is sorted in increasing order, then we have that checking the FPR and BPR takes at most $O(n \log_2(n))$ time (see Section 2).

Algorithm 3 differs from Algorithm 1 in two aspects. First, it can indicate for which reason the candidate set is not a valid period set if the check fails. Second, it also computes the set of "dying" period sets of Γ_{n-1}, that is the period set that do not remain valid at length n, nor cannot be extended at length n. We will define these notions in Section 5. Of course, dying period sets could also be computed within Algorithm 1, which uses the predicate Ξ (but for simplicity and to avoid redundancy, was not mentioned earlier).

Altogether the time complexity of Algorithm 3 is bounded by $O(n \log_2(n) \times \kappa_n)$, which may not be optimal.

▶ **Definition 14.** *A dying period set P is a period set of Γ_{n-1} such that neither P nor $P \cup \{n-1\}$ belong to Γ_n. In other words, P has no extension in Γ_n.*

Algorithm 3 IncrementalGamma with rule based certification

```
Input : n > 1: integer; Γ_{n-1}: the set of period sets for length n − 1
Output: Γ_n: the set of period sets for length n; D: the set of dying PS at length n;
G := ∅;                                   // G:  variable to store Γ_n
D := ∅;                                   // D:  variable to store dying PS
for  P ∈ Γ_{n-1} do
    Q := P ∪ {n − 1}        // build extension of P with period n − 1;
    if checkFPR( P, n ) then // n − 1 is required by FPR at length n
        if  checkBPR( Q, n ) then  insert Q in G;
        else  insert P in D              // otherwise P is dying at length n;
    else
        if  checkBPR( P, n ) then  insert P in G ;
        else
            if  checkBPR( Q, n ) then  insert Q in G ;
            else  insert P in D          // otherwise P is dying at length n;
return G and D;
```

C Algorithm Binary realization

C.1 Correctness and complexity of the algorithm

Proof. Let us prove that the Algorithm Binary Realization is correct.

Correction of the base case As we process the last period of P, the nested set is $\{0\}$ for length $n - max(P)$. We must build a suffix without period (i.e., whose basic period is its length). Hence, the word $\mathtt{a}.\mathtt{b}^{(\mathrm{prevLg}-1)}$ is a binary realization for this set.

Correction of the general case. After setting variables *lg* and *innerPeriod*, we check the condition ($innerPeriod < prevIP$). In a period set, the offset $P[i+1] - P[i]$ decreases when i increases. The condition implies the current nested set is invalid, and we return ϵ as needed. Another way to formulate this: If the condition is satisfied, then *suffix*, which ends with *prevSuffix*, does not satisfy the FPR, meaning that this set is invalid.

The invariant at the start of the for loop is that *prevSuffix* realizes the nested set $P_{P[i+1]}$ and has *prevIP* as basic period. By construction, we know that $lg = prevLg + innerPeriod$. By construction, *suffix* ends with *prevSuffix* and has basic period *innerPeriod*. Thus, by the invariant, *suffix* will realize $P_{P[i]}$.

Case 1 We build *suffix* by concatenating a prefix of *prevSuffix* of length *innerPeriod* with *prevSuffix* (line 10), and we must ensure that *suffix* has basic period *innerPeriod*. Let us consider the conditions from line 9.

1. If ($innerPeriod = prevIP$) then, as *prevSuffix* already has period *prevIP*, *suffix* will inherit from it. Otherwise we know that ($innerPeriod > prevIP$).
2. Then, *prevSuffix* has a basic period (*prevIP*) that should not divide *innerPeriod*, which is the length of the prefix of *prevSuffix* that occurs as prefix of *suffix*. Hence, we require the condition ($prevIP \nmid innerPeriod$) to be satisfied. Otherwise, *suffix* would also have *prevIP* as period; then *suffix* would be a binary world, but would not realize P.

3. Then, if ($innerPeriod = prevLg$) then $lg = 2\times$ $prevLg$ and *suffix* equals /prevSuffix/^{2} and has the desired length and basic period.
4. Otherwise, we check that *prevSuffix* has period *innerPeriod*. If yes, then *suffix* also has period *innerPeriod* by construction (line 10), and thus realizes $P_{P[i]}$. If not, then there is no possible realization of P and we return ϵ (line 11).

Case 2 Here, we know that *lg* is larger than twice *prevLg*. Therefore, we will build a prefix that starts with *prevSuffix* followed by *nb* new symbols, such that *suffix* has no period shorter than *innerPeriod*. Hence, we must ensure that *newPrefix* is primitive, otherwise it would have a period that divides *innerPeriod*. By Lemma 3 from [10], for any binary word w, w`a` or w`b` is primitive. So, we concatenate a^{nb} to *prevSuffix*, and check if it is primitive (in $O(|\text{newPrefix}|)$ time). If not, we change its last symbol by a `b`. In both cases, *newPrefix* is primitive. By construction, *suffix* has basic period *innerPeriod* as desired, and thus realizes $P_{P[i]}$. ◀

C.2 Examples of traces of binary realizations

Here is the trace of Algorithm 2 for length $n = 10$, and $P = \{0, 3, 6, 8\}$, which is not a valid period set for that length.

period	length	inner period	case	suffix	valid
8	10-8 = 2	10-8= 2	2	$z = ab$	true
6	10-6 = 4	8-6 = 2	2	$y = zz = abab$	true
3	10-3 = 7	6-3 = 3	1	$w = y \oplus_3 y$	false

The table below illustrates that the merge attempted at the last loop iteration for $P[i] = 3$ is impossible, since a mismatch occurs in the overlap.

pos.	0	1	2	3	4	5	6
y	a	b	a	b	-	-	-
y	-	-	-	a	b	a	b

D Checking FPR

Let us state some properties:

1. From the definition of FPR, we can see that checking the FPR for a pair (p, q) of P is equivalent to checking the FPR for pair $(0, q)$ in the nested PS P_p.
2. Assume the FPR is satisfied for pair $(0, p)$. Then, it is also satisfied for any pair (hp, jp) with $1 \leq h < j < \lfloor n/p \rfloor$ and $hp,jp \in P$, since both periods are multiples of p.

From both properties, we get that once the FPR has been checked for the first pair (p, q) taken that has offset $(q - p)$, it is also satisfied for any other pair whose offset equals r or a multiple of r. It follows that, for a set P, one can limit the checking of FPR only to left most pairs whose offsets differ from eachother and are not multiple of another offset. Thus, at least one element, say p, must be an *irreducible period* (as defined in [24]), and q is the closest period to p (i.e., one which gives rise to the smallest offset with respect to p). Since, the number of irreducible periods of a period set of Γ_n is bounded by $\log_2(n)$ [25], the number of such pairs also is. We obtain the bound on the complexity for the general case stated in Lemma 8.

E Fate: computation of the limits of a period set

E.1 Extension limit

Algorithm 4 computes the extension limit of P. The extension limit is a length at which some deducible period needs to be added to P to satisfy the FPR. It equals the added period plus one, and must be larger than the birth length of P (Indeed, P is a valid period for length at which is first occurs, and thus satisfies the FPR for that length). By definition of the FPR, a period induced by the FPR equals $P[i] + P[i] - P[j]$ for some indexes $0 < j < i < card(P)$. Because, we need the minimum of added periods, we can restrict the computation to pairs of adjacent periods (i.e. that is to case where $j = i - 1$), since the offset between periods decreases with their index. Hence, the formula $P[i] + (P[i] - P[i-1])$ for computing the limit induced from current period $P[i]$. Because of this, we can also rule out cases where $P[i]$ is smaller the half the birth length of P (line 6).

Algorithm 4 ExtensionLimit

Input : P: a valid period set (as an ordered list of integers)
Output: the extension limit of P (a minimum length at which P requires an extension);

1 birthLg $:= max(P) + 1$; `// min length` x `at which` P `first occurs in` $\Gamma(x)$
2 limit $:= max(int)$; `// limit to be computed, init. with largest integer`
3 **for** $i := card(P) - 1$ *to* 1 **do**
4   **if** $P[i] \leq \lfloor \frac{birthLg}{2} \rfloor$ **then** `//`
5   break; `// avoid such` $P[i]$ `values whose limit cannot be` $>$ `birthLg`
6   **if** $P[i] + (P[i] - P[i-1]) \geq birthLg$ *)* **then** `// current limit is beyond birthLg`
        `// update limit with the min of limit and current limit`
7   limit $:= min(\text{limit}, P[i] + (P[i] - P[i-1]))$;
8 **return** $limit + 1$;

E.2 Recursive FW limit

We exhibit an algorithm to compute what we termed, the recursive FW limit of a PS P (see Algorithm 5). The FW theorem provides a way to compute a maximal length for any pair of distinct, non trivial periods such that one period is not a multiple of the other. For any p, q in P such that $0 < p < q < n$ and $p \nmid q$, we denote by $FW(p, q)$ the FW limit, that is $FW(p, q) := p + q - gcd(p, q)$. If $p \div q$ we assume that $FW(p, q) := max(int)$. First, the algorithm proceeds with two special cases: if all periods are multiple of the basic period, then it returns $max(int)$. Note this includes the case with basic period equals to one. If P contains only three periods, then it returns $FW(P[1], P[2])$.

Otherwise, it will compute the limit l and initializes with $max(int)$. It loops over P backwards, to consider longer and longer suffixes starting at a position with period of a word satisfying P, and builds a list Q of periods restricted to the current suffix. The periods in Q are those of P minus the starting position. It computes $FW(Q[1], Q[2])$ and takes the minimum between l and $P[i] + FW(Q[1], Q[2])$. After terminating the loop, it returns the limit l.

Algorithm 5 RecursiveFWLimit

Input : P: a valid period set (as an ordered list of integers)
Output: the minimum length at which a pair of periods of P requires a change of basic period (application of FW theorem);

1 **if** $(P[1] \mid P[i])$ *for all* $1 < i < card(P)$ **then** `// If basic period divides all other periods`
2   **return** $max(int)$;
3 **if** $card(P) = 3$ **then** `// If` P `contains only two non trivial periods`
4   **return** $FW(P[1], P[2])$;
5 limit $:= max(int)$; `// limit to be computed, init. with largest integer`
6 insert $(P[n-1] - P[n-2])$ in Q ; `// Init` Q `with the last offset between periods`
7 **for** $i := card(P) - 3$ *to* 0 **do**
8   offset $:= P[i+1] - P[i]$;
9     $Q[0] := Q[0] +$ offset;
10   insert offset at first position in Q;
11   limit $:= min($limit$, P[i] + FW(Q[0], Q[1]))$;
12 **return** *limit*;

Complexity. In Algorithm 5, the first special case is processed in $card(P)$ time (lines 1–2), while the second one requires constant time (lines 3–4). The main loop is executed at most $card(P)$ times and all instructions in it take constant time (lines 7–11). Altogether, Algorithm 5 takes $O(card(P))$ time and constant space.

Correctness. The correctness of Algorithm 5 follows from Lemma 7.

F Properties of periods and characterization of period sets

F.1 Properties of periods

Let us state some known, useful properties of periods, which are detailed in [25].

▶ **Lemma 15.** *Let p be a period of $u \in \Sigma^n$ and $k \in \mathbb{N}_{\geq 0}$ such that $kp < n$. Then kp is also a period of u.*

▶ **Lemma 16.** *Let p be a period of $u \in \Sigma^n$ and q a period of the suffix $w = u[p\,..\,n-1]$. Then $(p+q)$ is a period of u. Moreover, $(p+kq)$ is also a period of u for all $k \in \mathbb{N}_{\geq 0}$ with $p + kq < n$.*

▶ **Lemma 17.** *Let p, q be periods of $u \in \Sigma^n$ with $0 \leq q \leq p$. Then the prefix and the suffix of length $(n-q)$ have the period $(p-q)$.*

▶ **Lemma 18.** *Suppose p is a period of $u \in \Sigma^n$ and there exists a substring v of u of length at least p and with period r, where $r|p$. Then r is also a period of u.*

F.2 Characterization of autocorrelations/period sets [8]

Guibas and Odlyzko have provided two equivalent characterizations of period sets: one is given by predicate Ξ, the other is the rule based characterization given in Section 2.3. However, they manipulate period sets as binary vectors called *autocorrelation* (or sometimes correlation for short). Remind that an autocorrelation is a binary encoding in a binary string of length n of a period set of Γ_n. We recall in extenso the original predicate Ξ and then their Theorem 5.1, which states the equivalence of characterizations and the alphabet independence.

> *Predicate* Ξ: v satisfies Ξ iff $v_0 = 1$ and, if p is the basic period of v, one of the following conditions is satisfied:
>
> *Case* a: $p \leqslant \lfloor n/2 \rfloor$. Let $r := \text{mod}\,(n,p)$, $q := p + r$ and w the suffix of v of length q. Then for all j in $[1, n-q]$ $v_j = 1$ if $j = ip$ for some i, and $v_j = 0$ otherwise; and the following conditions hold:
>
> 1. $r = 0$ or $w_p = 1$,
> 2. if $\pi(w) < p$ then $\pi(w) + p > q + \gcd(\pi(w), p)$,
> 3. w satisfies predicate Ξ.
>
> *Case* b: $p > \lfloor n/2 \rfloor$. We have $\forall j$: $1 \leqslant j < p$, $v_j = 0$. Let w be the suffix of v of length $n - p$, then w satisfies predicate Ξ.

▶ **Theorem 19.** *Let v a binary string of length n. The following statements are equivalent:*
1. *v is the autocorrelation of a binary word*
2. *v is the autocorrelation of a word over an alphabet of size ≥ 2*
3. *$v_0 = 1$ and v satisfies the **Forward** and **Backward Propagation Rules***
4. *v satisfies the predicate Ξ.*

Let $v \in \{0,1\}^n$. We state the original definitions of FPR and BPR.

▶ **Definition 20.** *v satisfies the FPR iff for all pairs (p,q) satisfying $0 \leq p < q < n$ and $v_p = v_q = 1$, it follows that $v_{p+i(q-p)} = 1$ for all $i = 2, \ldots, \lfloor (n-p)/(q-p) \rfloor$.*

▶ **Definition 21.** *v satisfies the BPR iff for all pairs (p,q) satisfying $0 \leq p < q < 2p$, $v_p = v_q = 1$, and $v_{2p-q} = 0$, it follows that $v_{p-i(q-p)} = 0$ for all $i = 2, \ldots, \min(\lfloor p/(q-p) \rfloor, \lfloor (n-p)/(q-p) \rfloor)$.*