



LABORATOIRE



INFORMATIQUE, SIGNAUX ET SYSTÈMES
DE SOPHIA ANTIPOLIS
UMR 6070

COMPUTING ABELIAN PERIODS IN WORDS

Gabriele Fici, Thierry Lecroq, Arnaud Lefebvre, Elise Prieur-Gaston

Equipe MC3

Rapport de recherche
ISRN I3S/RR-2011-01-FR

Janvier 2011

Computing Abelian Periods in Words

Gabriele Fici¹, Thierry Lecroq², Arnaud Lefebvre², and Élise Prieur-Gaston²

¹ `Gabriele.Fici@unice.fr`, I3S, CNRS/Université de Nice-Sophia Antipolis, France
² `{Thierry.Lecroq,Arnaud.Lefebvre,Elise.Prieur}@univ-rouen.fr`, LITIS
EA4108, Université de Rouen, 76821 Mont-Saint-Aignan Cedex, France

Abstract. In the last couple of years many works have been devoted to Abelian complexity of words. Recently, Constantinescu and Ilie introduced the notion of *Abelian period*. However, to the best of our knowledge, no algorithm is known for computing these periods. We show that a word of length n over an alphabet of size σ can have $O(n^2)$ distinct Abelian periods. We then present two off-line and three on-line algorithms for computing all the Abelian periods of a given word. The first off-line algorithm is a brute-force one, while the second uses a *select* array. The first on-line algorithm uses a two dimensional array, the second uses lists and the third uses heaps to process the periods by groups. The space complexity of the algorithms is $O(n^2)$. The time complexity of the first two algorithms is $O(n^3 \times \sigma)$ while the time complexity of the latter algorithm is $O(n^2 \times (n \log n) \times \sigma)$. Experimental results show that the off-line algorithm using the *select* array is the fastest in practice.

1 Introduction

An integer $p > 0$ is a (classical) period of a word w of length n if $w[i] = w[i + p]$ for any $1 \leq i \leq n - p + 1$. Classical periods have been extensively studied in combinatorics on words [13] due to their direct applications in data compression and pattern matching.

The Parikh vector of a word w enumerates the cardinality of each letter of the alphabet in w . The reader can refer to [5] for a list of applications of Parikh vectors.

An integer p is an Abelian period of a word w if w can be factorized as $u_0 u_1 \cdots u_{k-1} u_k$ where all the u_i 's are of length p and have the same Parikh vector \mathcal{P} for $0 < i < k$ and the Parikh vectors of u_0 and u_k are contained in \mathcal{P} [8]. This definition matches the one of *weak repetition* (also called *Abelian power*) when u_0 and u_k are the empty word and $k > 2$ [9].

In the last couple of years many works have been related to Abelian complexity [10, 1, 6, 3, 11, 2, 4, 14]. Efficient algorithms for Abelian pattern matching have been designed [12, 7]. However, apart of the greedy off-line algorithm given in [9], neither efficient nor on-line algorithm is known for computing all the Abelian periods of a given word.

In this article we present different and efficient off-line algorithm and different on-line algorithms for computing all the Abelian periods of a given word. In

Sec. 2 we give some basic definitions and notations. Sec. 3 presents a greedy off-line algorithm while Sec. 4 presents our three on-line algorithms. In Sec. 5 we give some experimental results on execution times and average Abelian periods lengths. Finally, Sec. 6 contains conclusions and perspectives.

2 Definitions and notation

Let $\Sigma = \{a_1, a_2, \dots, a_\sigma\}$ be a finite alphabet of cardinality σ . We consider words $w \in \Sigma^*$. We denote by $|w| = n$ the length of w and write $w[i]$ the i -th symbol of w , $1 \leq i \leq |w|$. We denote by $|w|_a$ the number of occurrences of the symbol $a \in \Sigma$ in the word w .

The Parikh vector of a word w , denoted by $\mathcal{P}(w)$, enumerates the cardinality of each letter of Σ in w . Notice that two words have the same Parikh vector if and only if one is a permutation of the other (*i.e.*, one is the anagram of the other).

We denote by $\mathcal{P}_w(i, m)$ the Parikh vector of the factor of length m beginning at position i in the word w .

Given a Parikh vector \mathcal{P} , we denote by $\mathcal{P}[i]$ its i -th component, and by $|\mathcal{P}|$ the sum of its components, that is $|\mathcal{P}| = \sum_{i=1}^{\sigma} \mathcal{P}[i]$. Given two Parikh vectors \mathcal{P}, \mathcal{Q} , we write $\mathcal{P} \subset \mathcal{Q}$ if $\mathcal{P}[i] \leq \mathcal{Q}[i]$ for every $1 \leq i \leq \sigma$ and $|\mathcal{P}| < |\mathcal{Q}|$.

Definition 1 ([8]). *A word w has an Abelian period (h, p) if $w = u_0 u_1 \cdots u_{k-1} u_k$ such that:*

- $\mathcal{P}(u_0) \subset \mathcal{P}(u_1) = \dots = \mathcal{P}(u_{k-1}) \supset \mathcal{P}(u_k)$,
- $|\mathcal{P}(u_0)| = h$, $|\mathcal{P}(u_1)| = p$.

We call u_0 and u_k resp. the *head* and the *tail* of the Abelian period. Notice that the length of the tail $t = |u_k|$ is uniquely determined by h, p and $|w|$ namely $t = (|w| - h) \bmod p$.

The following lemma gives a bound on the maximal number of Abelian periods of a word.

Lemma 2. *The maximal number of Abelian periods of a word of length n is $O(n^2)$.*

Proof. The word a^n , $a \in \Sigma$, has Abelian period (h, p) for every $h < p$, $h + p \leq n$.

A natural order can be defined on the Abelian periods.

Definition 3. *Two distinct Abelian periods (h, p) and (h', p') of a word w are ordered as follows: $(h, p) < (h', p')$ if $p < p'$ or $(p = p'$ and $h < h')$.*

We are interested in computing all the Abelian periods of a word.

3 Off-line algorithms

3.1 Brute-force algorithm

In Fig. 1 we present a brute-force algorithm which computes all the Abelian periods of an input word w of length n . For each possible head of length h , from 1 to $\lfloor (n-1)/2 \rfloor$ the algorithm tests all the possible values p such that $p > h$ and $h+p \leq n$. This is a reformulation of the algorithm given in [9]. The algorithm easily adapts to give only the smallest Abelian period or the weak repetitions.

```

BRUTEFORCE( $w, n$ )
1  for  $p \leftarrow 1$  to  $n$  do
2     $h \leftarrow 0$ 
3    while  $h + p \leq n$  do
4      if  $(h, p)$  is an Abelian period of  $w$  then
5        OUTPUT( $h, p$ )
6       $h \leftarrow h + 1$ 

```

Fig. 1. Brute-force algorithm for computing all the Abelian periods of a word w of length n .

Example 4. For $w = \text{abaababa}$ the algorithm outputs $(1, 2)$, $(0, 3)$, $(2, 3)$, $(1, 4)$, $(2, 4)$, $(3, 4)$, $(0, 5)$, $(1, 5)$, $(2, 5)$, $(3, 5)$, $(0, 6)$, $(1, 6)$, $(2, 6)$, $(0, 7)$, $(1, 7)$ and $(0, 8)$. Among these periods $(1, 2)$ is the smallest and $(1, 2)$, $(0, 3)$ and $(2, 3)$ correspond to the weak repetitions $\text{ba} \cdot \text{ab} \cdot \text{ab}$, $\text{aba} \cdot \text{aba}$ and $\text{aab} \cdot \text{aba}$ respectively.

Theorem 5. *The algorithm BRUTEFORCE computes all the Abelian periods of a given word of length n in time $O(n^2 \times \sigma)$.*

Proof. The correctness of the algorithm comes directly from Def. 1. The test in line 4 is performed $O(n^2)$ times. Each test in line 4 consists in comparing n/p Parikh vectors. Comparing two Parikh vectors can be done in $\Theta(\sigma)$ time. Thus the total time complexity of algorithm BRUTEFORCE is $O(n^2 \times \sigma)$.

3.2 Select-based algorithm

Let introduce the *select* array defined as follows.

Definition 6. *Let w be a word of length n over alphabet Σ , then $\forall a \in \Sigma$:*

- $\text{select}[a, 0] = 0$;
- $\forall 1 \leq i \leq |w|_a$, $\text{select}[a, i] = j$ iff j is the position of the i -th occurrence of letter a in w ;
- $\forall i > |w|_a$, $\text{select}[a, i] = +\infty$.

Example 7. For $w = \text{abaababa}$ the *select* array is:

	0	1	2	3	4	5
a	0	1	3	4	6	8
b	0	2	5	7	$+\infty$	$+\infty$

Clearly, the computation of the *select* array of a word w of length n can be done in one pass over w and takes $O(n)$ time and space (with constant 1). This corresponds to the prefix inverted table in [7].

The brute-force algorithm tests all possible pairs (h, p) but it is clear that, given h , some pairs cannot be Abelian periods. For example, let $w = \text{abaaaaabaa}$ and $h = 2$. Since $\mathcal{P}_w(1, h)$ has to be included in $\mathcal{P}_w(h + 1, p)$, the couples $(2, 3)$, $(2, 4)$ and $(2, 5)$ can not be Abelian periods of w : the minimal p value such that $(2, p)$ can be an Abelian period is in fact 6, in order to include the second **b** of w . This remark leads to the following definitions and propositions.

Definition 8. Let w be a word of length n on alphabet Σ . Then $\forall 0 \leq h \leq \lfloor (n - 1)/2 \rfloor$, $\mathcal{M}[h]$ is defined by $\mathcal{M}[h] = \max\{\text{select}[a, 2 \times |w[1..h]|_a] \mid a \in \Sigma\}$

Proposition 9. Let w be a word of length n on alphabet Σ . Let $0 \leq h \leq \lfloor (n - 1)/2 \rfloor$. Then one of the three cases holds:

1. if $\mathcal{M}[h] > n$ then no p exists such that (h, p) is an Abelian period of w ;
2. if $\mathcal{M}[h] = 2 \times h$ then $h + 1$ is the minimal p value such that $\mathcal{P}_w(1, h) \subset \mathcal{P}_w(h + 1, p)$;
3. if $2 \times h + 1 \leq \mathcal{M}[h] \leq n$ then $\mathcal{M}[h] - h$ is the minimal p value such that $\mathcal{P}_w(1, h) \subset \mathcal{P}_w(h + 1, p)$.

Proof. These three cases are exclusive and are the only possible cases. Case 1 means that $\exists a \in \Sigma$ such that $|w[1..h]|_a > |w[h + 1..n]|_a$: w has no Abelian period with head length h , and then case 1 holds. Cases 2 and 3 mean that there exists a factor $w[h + 1.. \mathcal{M}[h]]$ such that $\mathcal{P}_w(1, h) \subseteq \mathcal{P}_w(h + 1, \mathcal{M}[h] - h)$. Case 2 corresponds to $\mathcal{M}[h] = 2 \times h$, then $\mathcal{P}_w(1, h) = \mathcal{P}_w(h + 1, \mathcal{M}[h] - h)$. According to Def. 1, the minimal p value such that $\mathcal{P}_w(1, h) \subset \mathcal{P}_w(h + 1, p)$ is $h + 1$: case 2 holds. Case 3 means that $\mathcal{P}_w(1, h) \subset \mathcal{P}_w(h + 1, \mathcal{M}[h] - h)$. Consider letter $a \in \Sigma$ such that $\mathcal{M}[h] = \text{select}[a, 2 \times |w[1..h]|_a]$. Consider $0 < m < \mathcal{M}[h]$. Then $|w[h + 1..m]|_a < |w[h + 1.. \mathcal{M}[h]|_a = |w[1..h]|_a$. Then $\mathcal{P}_w(h + 1, h)$ can not be included in $\mathcal{P}_w(h + 1, m - h)$. Consequently case 3 holds.

Consider now the following definition.

Definition 10. Let w be a word of length n on alphabet Σ . Then $\forall 0 \leq h \leq \lfloor (n - 1)/2 \rfloor$, $\mathcal{G}[h]$ is defined by

$$\mathcal{G}[h] = \max\{ \text{select}[a, i + 1] - \text{select}[a, i] \mid a \in \Sigma, \\ h < \text{select}[a, i] < \text{select}[a, i + 1] \leq n \}$$

This array can be computed in linear time, processing positions from the end of w , using an extra array of size σ .

Proposition 11. *Let w be a word of length n on alphabet Σ . Let $0 \leq h \leq \lfloor (n-1)/2 \rfloor$. If $h < p < \max(\mathcal{M}[h] - h, \lfloor (\mathcal{G}[h] + 1)/2 \rfloor)$ then (h, p) is not an Abelian period of w .*

Proof. It is a direct consequence of Prop. 9 and Def. 10.

Using Prop. 11, algorithm OFF-LINE-SELECT, given in Fig. 2, improves the brute-force algorithm.

```

OFF-LINE-SELECT( $w, n$ )
1   $Select \leftarrow \text{COMPUTeselect}(w, n)$ 
2   $\mathcal{G} \leftarrow \text{COMPUTEG}(w, n)$ 
3   $h \leftarrow 0$ 
4  while  $h < \lfloor (n-1)/2 \rfloor$  do
5     $\mathcal{M}[h] \leftarrow \text{COMPUTEM}(h, Select)$ 
6     $p \leftarrow \max(\mathcal{M}[h] - h, \lfloor (\mathcal{G}[h] + 1)/2 \rfloor)$ 
7    if  $p = h$  then
8       $p \leftarrow p + 1$ 
9    while  $h + p \leq n$  do
10     if  $\text{SHIFT}(h, p, Select) = \text{ok}$  then
11        $\text{OUTPUT}(h, p)$ 
12      $p \leftarrow p + 1$ 
13    $h \leftarrow h + 1$ 

```

Fig. 2. Algorithm computing all the Abelian periods of word w of length n , based on the *select* array.

Proposition 12. *Algorithm OFF-LINE-SELECT computes all the Abelian periods of word w of length n in time $O(n^2 \times \sigma)$ and space $O(n^2 + \sigma)$.*

Proof. The *select* and \mathcal{G} arrays can be computed in $O(n)$ time and space. For each value h , according to Def. 8, $\mathcal{M}[h]$ value can be computed in $O(\sigma)$. According to Prop. 11, value p computed in line 6 is the minimal value such that (h, p) can be an Abelian period of w . The SHIFT function, line 10, simply verifies that (h, p) is an Abelian period of w . Consequently, algorithm OFF-LINE-SELECT computes all the Abelian periods of w in time $O(n^2 \times \sigma)$ and space $O(n^2 + \sigma)$.

4 On-line algorithms

We now propose three on-line algorithms to compute all the Abelian periods of a word w using dynamic programming. When processing $w[i]$, in the first algorithm, using a two dimensional array, we inspect all the possible values (h, p) ; in the second algorithm, using lists, we inspect only the Abelian periods

of $w[1..i-1]$; in the third one, using heaps, we inspect the Abelian periods of $w[1..i-1]$ by groups built depending upon the tail length of the periods.

The following proposition states that if (h, p) is not an Abelian period of a word w , with $h + p \leq n = |w|$, then it cannot be an Abelian period of any word having w as prefix.

Proposition 13. *Let w be a word of length n and let h, p such that $h + p \leq n$. If (h, p) is not an Abelian period of w , then (h, p) is not an Abelian period of wa for any symbol $a \in \Sigma$.*

Proof. If (h, p) is not an Abelian period of w , at least one of the following three cases holds:

1. $\mathcal{P}_w(1, h) \not\subseteq \mathcal{P}_w(h+1, p)$;
2. there exist two distinct indices $h \leq i, i' \leq |w| - p + 1$ such that $i = kp + h + 1$ and $i' = k'p + h + 1$ with k and k' two integers and $\mathcal{P}_w(i, p) \neq \mathcal{P}_w(i', p)$;
3. $t = (|w| - h) \bmod p$ and $\mathcal{P}_w(|w| - t + 1, t) \not\subseteq \mathcal{P}_w(|w| - p - t + 1, p)$.

If case 1 holds then $\mathcal{P}_{wa}(1, h) \not\subseteq \mathcal{P}_{wa}(h+1, p)$ and (h, p) is not an Abelian period of wa . If case 2 holds then $\mathcal{P}_{wa}(i, p) \neq \mathcal{P}_{wa}(i', p)$ and (h, p) is not an Abelian period of wa . If case 3 holds then $\mathcal{P}_{wa}(|w| - t + 1, t) \not\subseteq \mathcal{P}_{wa}(|w| - p - t + 1, p)$ and (h, p) is not an Abelian period of wa .

4.1 Two dimensional array

This algorithm uses a two dimensional array and Prop. 13 to compute all the Abelian periods of an input word w in an on-line manner. It processes the positions of w in increasing order. When processing position i , $T[h, p] = j$ iff $w[1..j]$ is the longest prefix of $w[1..i]$ having Abelian period (h, p) . Thus if $j = i - 1$ the algorithm checks whether $w[1..i]$ has Abelian period (h, p) and updates $T[h, p]$ accordingly.

When $T[h, p] = n$ it means that $w[1..n]$ is the longest prefix of w that has (h, p) as an Abelian period. Thus when $T[h, p] = n$ it means that (h, p) is an Abelian period of w .

Example 14. For $w = \text{abaababa}$ the algorithm computes the following array T :

$h \backslash p$	1	2	3	4	5	6	7	8
0	1	3	8	6	8	8	8	8
1		8	6	8	8	8	8	
2			8	8	8	8		
3				8	8			

Cells $T[h, p] = 8$ correspond to pairs output by algorithm BRUTEFORCE of example 4. Empty cells on the left part of the array correspond to cases where $h \geq p$ and empty cells on the right part correspond to cases where $h + p > 8$.

4.2 Lists

This algorithm also processes the position of w in increasing order. When processing position i it only stores pairs (h, p) such that $w[1..i]$ has Abelian period (h, p) : these pairs correspond to all the cells of array T , computed by the previous algorithm, such that $T[h, p] = i$.

At the end of this process, when $i = n$, this list contains all the Abelian periods of w , and only them.

4.3 Heaps

The following proposition shows that the set of Abelian periods of a prefix of a word can be partitioned into subsets depending of the length of the tail. In some cases all the periods of a subset can all be processed at once by inspecting only the smallest period of the subset.

Proposition 15. *Let w have s Abelian periods $(h_1, p_1) < (h_2, p_2) < \dots < (h_s, p_s)$ such that $(|w| - h_i) \bmod p_i = t > 0$ for $1 \leq i \leq s$. If (h_1, p_1) is an Abelian period of wa for any symbol $a \in \Sigma$ then $(h_2, p_2), \dots, (h_s, p_s)$ are also Abelian periods of wa .*

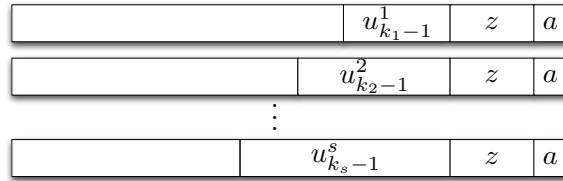


Fig. 3. $w = u_0^i u_1^i \dots u_{k_i-1}^i u_{k_i}^i$, $u_{k_i}^i = z$ for $1 \leq i \leq s$. If $\mathcal{P}(za) \subseteq \mathcal{P}(u_{k_1-1}^1)$ then $\mathcal{P}(za) \subseteq \mathcal{P}(u_{k_i-1}^i)$ for every $2 \leq i \leq s$.

Proof. Since $(h_1, p_1) < (h_2, p_2) < \dots < (h_s, p_s)$ are Abelian periods of w , $w = u_0^i u_1^i \dots u_{k_i-1}^i u_{k_i}^i$ with $|u_0^i| = h_i$, $|u_j^i| = p_i$ and $|u_{k_i}^i| = t$ for $1 \leq i \leq s$ and $1 \leq j \leq k_i$. If (h_1, p_1) is an Abelian period of wa , $\mathcal{P}(u_{k_1}^1 a) \subseteq \mathcal{P}(u_{k_1-1}^1)$. Since $|u_{k_1}^1| = |u_{k_i}^i|$ and $|u_{k_1-1}^1| \leq |u_{k_i-1}^i|$ we have that $\mathcal{P}(u_{k_i}^i a) \subseteq \mathcal{P}(u_{k_i-1}^i)$ for $2 \leq i \leq s$. Thus $(h_2, p_2), \dots, (h_s, p_s)$ are Abelian periods of wa . (See Fig. 3.)

The algorithm given in Fig. 4 uses Prop. 15 for computing all the Abelian periods by gathering all the ongoing periods (h, p) with the same tail length together in a heap where the element at the root of the heap is the smallest period.

When processing $w[i]$, the algorithm processes every heap H for the different tail lengths:

- if the period (h, p) at the root of H is a period of $w[1..i]$ then by Prop. 15 all the elements of H are Abelian periods of $w[1..i]$. If the tail length becomes equal to p then (h, p) is removed from the current heap and is moved into a new heap corresponding to the empty tail.
- if the period (h, p) at the root of H is not a period of $w[1..i]$ then it is removed from H and the same process is applied until a pair (h', p') is an Abelian period of $w[1..i]$ or the heap becomes empty. In the former case, by Prop. 15, all the remaining elements of H are Abelian periods of $w[1..i]$. This is realized by function EXTRACTUNTILOK in Line 11.

Then all the degenerate cases (h, p) such that $h + p = i$ have to be inserted in the heap corresponding to the empty tail (lines 15 to 18).

The function $\text{ROOT}(H)$ returns the smallest element of the heap H , the function $\text{INSERT}(H, e)$ inserts element e in the heap H , while the function $\text{REMOVE}(H)$ removes the smallest element of the heap H .

```

ABELIANPERIOD-HEAP( $w, n$ )
1   $L \leftarrow$  list with one heap containing  $(0, 1)$ 
2  for  $i \leftarrow 2$  to  $n$  do
3    NewHeap  $\leftarrow \emptyset$ 
4    for all  $H \in L$  do
5       $(h, p) \leftarrow \text{ROOT}(H)$ 
6       $d \leftarrow (i - h) \bmod p$ 
7      if  $d = 0$  then
8         $t \leftarrow p$ 
9      else  $t \leftarrow d$ 
10     if  $\mathcal{P}_w(i - t + 1, t) \not\subseteq \mathcal{P}_w(i - t - p + 1, p)$  then
11       EXTRACTUNTILOK( $H$ )
12     else if  $d = 0$  then
13       REMOVE( $H$ )
14       INSERT(NewHeap,  $(h, p)$ )
15      $h \leftarrow 0$ 
16     while  $h < \lfloor (i + 1)/2 \rfloor$  et  $\mathcal{P}_w(1, h) \subset \mathcal{P}_w(h + 1, i - h)$  do
17       INSERT(NewHeap,  $(h, i - h)$ )
18        $h \leftarrow h + 1$ 
19      $L \leftarrow L \cup \text{NewHeap}$ 
20 return  $L$ 

```

Fig. 4. On-line algorithm for computing all the Abelian periods of a word w of length n using heaps.

Theorem 16. *The algorithm ABELIANPERIOD-HEAP computes all the Abelian periods of a given word of length n in time $O(n^2 \times (n \log n) \times \sigma)$ and space $O(n^2)$.*

Proof. The correctness of the algorithm comes from Prop. 15. The maximal number of heaps is $n/2$ and the total number of elements of all the heaps is $O(n^2)$ (Lemma 2). The space complexity for the list L is $O(n^2)$. The time complexity of the algorithm is due to the two **for** loops of lines 2 and 4 and the different calls to EXTRACTUNTILOK in line 11 and INSERT and REMOVE. The maximal number of heaps is $n/2$, and the maximal number of elements in a single heap is n . Thus, the total complexity for the calls to EXTRACTUNTILOK, INSERT and REMOVE in a single run of the **for** loop of line 4 is $O(n \log n)$.

5 Experimental results

To compare practical performances of the different algorithms, they have been implemented in C in a homogeneous way and ran on test sets of random words (1000 words each) of different lengths (from 10 to 500) on different alphabet sizes (2, 4 and 8).

Tests were performed on a computer running Mac OS X with a 2.2 Ghz processor and 2 Go RAM.

Fig. 5 presents average running times of the two fastest algorithms: the off-line using *select* array and the on-line using heaps. The results show that, as expected, the off-line algorithm using *select* array is much more faster than the other ones. Moreover, our tests show that even the on-line algorithm using heaps improves dramatically on the brute-force one. One can remark that the difference of running times between the two algorithms increases as the word length grows.

$\sigma \setminus w $	10		50		100		200		500	
	<i>Heaps</i>	<i>Select</i>	<i>Heaps</i>	<i>Select</i>	<i>Heaps</i>	<i>Select</i>	<i>Heaps</i>	<i>Select</i>	<i>Heaps</i>	<i>Select</i>
2	2	1	4	2	14	4	58	18	477	161
4	1	1	4	2	12	4	52	20	466	142
8	1	1	2	1	8	4	44	17	420	176

Fig. 5. Average running times (in ms) of the algorithms using heaps and *select* on alphabets of sizes 2, 4 and 8, on words of lengths 10, 50, 100, 200 and 500.

6 Conclusion and perspectives

In this paper we presented different algorithms to compute all the Abelian periods of a word.

Current works on Abelian periods properties are in progress in order to improve their complexities. One way would consist in reporting only pairwise non-reducible Abelian periods, which would reduce their number. It also remains to obtain a bound on the minimal Abelian period given a word length and an alphabet size.

Simple modifications of the presented algorithms would allow to compute the smallest Abelian period of each factor of a word. This could have practical applications in areas such as bioinformatics and more precisely in the detection of DNA regions of homogeneous compositions.

References

1. S.V. Avgustinovich, A. Glen, B.V. Halldórsson, and S. Kitaev. On shortest crucial words avoiding abelian powers. *Discrete Applied Mathematics*, 158(6):605–607, 2010.
2. S.V. Avgustinovich, J. Karhumäki, and S. Puzynina. On Abelian versions of Critical Factorization Theorem. In *Proceedings of the 13th Mons Theoretical Computer Science Days*, 2010.
3. F. Blanchet-Sadri, J. I. Kim, R. Mercas, W. Severa, and S. Simmons. Abelian square-free partial words. In A.-H. Dediu, H. Fernau, and C. Martín-Vide, editors, *Proceedings of the 4th International Conference Language and Automata Theory and Applications*, volume 6031 of *Lecture Notes in Computer Science*, pages 94–105. Springer, 2010.
4. F. Blanchet-Sadri, A. Tebbe, and A. Veprauskas. Fine and Wilf’s theorem for abelian periods in partial words. In *Proceedings of the 13th Mons Theoretical Computer Science Days*, 2010.
5. P. Burcsi, F. Cicalese, G. Fici, and Z. Lipták. On table arrangements, scrabble freaks, and jumbled pattern matching. In P. Boldi and L. Gargano, editors, *Fun with Algorithms, 5th International Conference, FUN 2010*, volume 6099 of *Lecture Notes in Computer Science*, pages 89–101. Springer, 2010.
6. J. Cassaigne, G. Richomme, K. Saari, and L.Q. Zamboni. Avoiding abelian powers in binary words with bounded abelian complexity. *CoRR*, abs/1005.2514, 2010.
7. F. Cicalese, G. Fici, and Z. Lipták. Searching for jumbled patterns in strings. In J. Holub and J. Zdárek, editors, *Proceedings of the Prague Stringology Conference 2009, Prague, Czech Republic*, pages 105–117. Prague Stringology Club, Department of Computer Science and Engineering, Faculty of Electrical Engineering, Czech Technical University in Prague, 2009.
8. S. Constantinescu and L. Ilie. Fine and Wilf’s theorem for abelian periods. *Bull. Europ. Assoc. Theor. Comput. Sci.*, 89:167–170, 2006.
9. L. J. Cummings and W. F. Smyth. Weak repetitions in strings. *J. Combinatorial Mathematics and Combinatorial Computing*, 24:33–48, 1997.
10. J. D. Currie and A. Aberkane. A cyclic binary morphism avoiding abelian fourth powers. *Theor. Comput. Sci.*, 410(1):44–52, 2009.
11. M. Domaratzki and N. Rampersad. Abelian primitive words. *CoRR*, abs/1006.4104, 2010.
12. T. Ejaz, S. Rahmann, and J. Stoye. On abelian pattern matching. Technical Report 2008–01, Technische Fakultät der Universität Bielefeld, 2008.
13. M. Lothaire. *Algebraic Combinatorics on Words*. Cambridge University Press, 2002.
14. A.V. Samsonov and A.M. Shur. On abelian repetition threshold. In *Proceedings of the 13th Mons Theoretical Computer Science Days*, 2010.