Quasi-linear Time Computation of the Abelian Periods of a Word

Gabriele Fici[1], Thierry Lecroq[2], Arnaud Lefebvre[2], Élise Prieur-Gaston[2], and William F. Smyth[3]

[1] I3S, CNRS & Université Nice Sophia Antipolis, France
Gabriele.Fici@unice.fr
[2] LITIS EA4108, Université de Rouen, 76821 Mont-Saint-Aignan Cedex, France
{Thierry.Lecroq,Arnaud.Lefebvre,Elise.Prieur}@univ-rouen.fr
[3] Department of Computing and Software,
McMaster University, Hamilton ON L8S 4K1, Canada
smyth@mcmaster.ca

Abstract. In the last couple of years many research papers have been devoted to Abelian complexity of words. Recently, Constantinescu and Ilie (Bulletin EATCS 89, 167–170, 2006) introduced the notion of *Abelian period*. In this article we present two quadratic brute force algorithms for computing Abelian periods for special cases and a quasi-linear algorithm for computing all the Abelian periods of a word.

Keywords: Abelian period, Abelian repetition, weak repetition, design of algorithms, text algorithms, combinatorics on words

1 Introduction

An integer $p > 0$ is a (classical) period of a word $\boldsymbol{w}$ of length n if $\boldsymbol{w}[i] = \boldsymbol{w}[i+p]$ for any $1 \leqslant i \leqslant n-p$. Classical periods have been extensively studied in combinatorics on words [16] due to their direct applications in data compression and pattern matching.

The Parikh vector of a word $\boldsymbol{w}$ enumerates the cardinality of each letter of the alphabet in $\boldsymbol{w}$. For example, given the alphabet $\Sigma = \{a, b, c\}$, the Parikh vector of the word $\boldsymbol{w} = aaba$ is $(3, 1, 0)$. The reader can refer to [6] for a list of applications of Parikh vectors.

An integer p is an *Abelian period* for a word $\boldsymbol{w}$ over a finite alphabet $\Sigma = \{a_1, a_2, \ldots, a_\sigma\}$ if $\boldsymbol{w}$ can be written as $\boldsymbol{w} = \boldsymbol{u}_0\boldsymbol{u}_1 \cdots \boldsymbol{u}_{k-1}\boldsymbol{u}_k$ where for $0 < i < k$ all the $\boldsymbol{u}_i$'s have the same Parikh vector $\mathcal{P}$ such that $\sum_{i=1}^{\sigma} \mathcal{P}[i] = p$ and the Parikh vectors of $\boldsymbol{u}_0$ and $\boldsymbol{u}_k$ are contained in $\mathcal{P}$ [11]. For example, the word $\boldsymbol{w} = ababbbabb$ can be written as $\boldsymbol{w} = \boldsymbol{u}_0\boldsymbol{u}_1\boldsymbol{u}_2\boldsymbol{u}_3$, with $\boldsymbol{u}_0 = a$, $\boldsymbol{u}_1 = bab$, $\boldsymbol{u}_2 = bba$ and $\boldsymbol{u}_3 = bb$, and 3 is an Abelian period of $\boldsymbol{w}$ with Parikh vector $(1, 2)$ over $\Sigma = \{a, b\}$.

This definition of Abelian period matches that of *weak repetition* (also called *Abelian power*) when $\boldsymbol{u}_0$ and $\boldsymbol{u}_k$ are the empty word and $k > 2$ [12].

In the last couple of years many research papers have been devoted to Abelian complexity [13,1,8,3,14,2,4,20]. Efficient algorithms for Abelian Pattern Matching (also known as Jumbled Pattern Matching) have been designed [10,5,6,17,18,7].

Recently [15] gave algorithms for computing all the Abelian periods of a word of length n in time $O(n^2 \times \sigma)$. This was improved to time $O(n^2)$ in [9].

In this article we present a quasi-linear time algorithm for computing the Abelian periods of a word. In Section 2 we give some basic definitions and notation. Section 3 presents brute force algorithms while Section 4 presents our main contribution. Finally, Section 5 contains conclusions and perspectives.

Gabriele Fici, Thierry Lecroq, Arnaud Lefebvre, Élise Prieur-Gaston, William F. Smyth: Quasi-linear Time Computation of the Abelian Periods of a Word, pp. 103–110.
Proceedings of PSC 2012, Jan Holub and Jan Žďárek (Eds.), ISBN 978-80-01-05095-8

2 Notation

Let $\Sigma = \{a_1, a_2, \ldots, a_\sigma\}$ be a finite ordered alphabet of cardinality σ and Σ^* the set of words on alphabet Σ. We denote by $|\boldsymbol{w}|$ the length of the word $\boldsymbol{w}$. We write $\boldsymbol{w}[i]$ for the i-th symbol of $\boldsymbol{w}$ and $\boldsymbol{w}[i \,..\, j]$ for the factor of $\boldsymbol{w}$ from the i-th symbol to the j-th symbol, with $1 \leqslant i \leqslant j \leqslant |\boldsymbol{w}|$. We denote by $|\boldsymbol{w}|_a$ the number of occurrences of the symbol $a \in \Sigma$ in the word $\boldsymbol{w}$.

The *Parikh vector* of a word $\boldsymbol{w}$, denoted by $\mathcal{P}_{\boldsymbol{w}}$, counts the occurrences of each letter of Σ in $\boldsymbol{w}$; that is $\mathcal{P}_{\boldsymbol{w}} = (|\boldsymbol{w}|_{a_1}, \ldots, |\boldsymbol{w}|_{a_\sigma})$. Notice that two words have the same Parikh vector if and only if one word is a permutation of the other.

Given the Parikh vector $\mathcal{P}_{\boldsymbol{w}}$ of a word $\boldsymbol{w}$, we denote by $\mathcal{P}_{\boldsymbol{w}}[i]$ its i-th component and by $|\mathcal{P}_{\boldsymbol{w}}|$ the sum of its components. Thus for $\boldsymbol{w} \in \Sigma^*$ and $1 \leqslant i \leqslant \sigma$, we have $\mathcal{P}_{\boldsymbol{w}}[i] = |\boldsymbol{w}|_{a_i}$ and $|\mathcal{P}_{\boldsymbol{w}}| = \sum_{i=1}^{\sigma} \mathcal{P}_{\boldsymbol{w}}[i] = |\boldsymbol{w}|$.

Finally, given two Parikh vectors $\mathcal{P}, \mathcal{Q}$, we write $\mathcal{P} \subset \mathcal{Q}$ if $\mathcal{P}[i] \leqslant \mathcal{Q}[i]$ for every $1 \leqslant i \leqslant \sigma$ and $|\mathcal{P}| < |\mathcal{Q}|$.

Definition 1 ([11]). *A word $\boldsymbol{w}$ has an Abelian period (h, p) if $\boldsymbol{w} = \boldsymbol{u}_0\boldsymbol{u}_1 \cdots \boldsymbol{u}_{k-1}\boldsymbol{u}_k$ such that:*

- $\mathcal{P}_{\boldsymbol{u}_0} \subset \mathcal{P}_{\boldsymbol{u}_1} = \cdots = \mathcal{P}_{\boldsymbol{u}_{k-1}} \supset \mathcal{P}_{\boldsymbol{u}_k}$,
- $|\mathcal{P}_{\boldsymbol{u}_0}| = h$, $|\mathcal{P}_{\boldsymbol{u}_1}| = p$.

We call $\boldsymbol{u}_0$ and $\boldsymbol{u}_k$ resp. the *head* and the *tail* of the Abelian period. Notice that the length $t = |\boldsymbol{u}_k|$ of the tail is uniquely determined by h, p and $|\boldsymbol{w}|$, namely $t = (|\boldsymbol{w}| - h) \bmod p$.

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

Lemma 2 ([15]). *The maximum number of Abelian periods for a word of length n over the alphabet Σ is $\Theta(n^2)$.*

Proof. The word $(a_1a_2 \cdots a_\sigma)^{n/\sigma}$ has Abelian period (h, p) for any $p \equiv 0 \bmod \sigma$ and $h < p$. □

A natural order can be defined on the Abelian periods.

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

Definition 4 ([9]). *Let $\boldsymbol{w}$ be a word of length n. Then the mapping $pr : \Sigma \to A$, where A is the set of the first σ prime numbers, is defined by:*

$$pr(\sigma_i) = i\text{-th prime number.}$$

The P-signature of $\boldsymbol{w}$ is defined by:

$$P\text{-}signature(\boldsymbol{w}) = \Pi_{i=1}^{n} pr(\boldsymbol{w}[i]).$$

Definition 5 ([9]). *Let $\boldsymbol{w}$ be a word of length n. Then the mapping $s : \Sigma \to B$, where B is the set of the first $\sigma - 1$ powers of $n + 1$ and 0, is defined by:*

$$s(\sigma_i) = \begin{cases} 0 & \text{if } i = 1 \\ (n+1)^{i-2} & \text{otherwise.} \end{cases}$$

The S-signature of $\boldsymbol{w}$ is defined by:

$$S\text{-}signature(\boldsymbol{w}) = \sum_{i=0}^{n} s(\boldsymbol{w}[i]).$$

Observation 1 ([9]) *For a word $\boldsymbol{w}$ of length n the array Pr of n elements is defined by*

$$Pr[i] = \Pi_{j=1}^{i} pr(\boldsymbol{w}[j]),$$

then

$$P\text{-}signature(\boldsymbol{w}[k\,..\,\ell]) = \begin{cases} Pr[\ell]/Pr[k-1] & \text{if } k \neq 0 \\ Pr[\ell] & \text{otherwise.} \end{cases}$$

Observation 2 ([9]) *For a word $\boldsymbol{w}$ of length n the array S of n elements is defined by*

$$S[i] = \sum_{j=1}^{i} s(\boldsymbol{w}[j]),$$

then

$$S\text{-}signature(\boldsymbol{w}[k\,..\,\ell]) = \begin{cases} S[\ell] - S[k-1] & \text{if } k \neq 0 \\ S[\ell] & \text{otherwise.} \end{cases}$$

Example 6. $\boldsymbol{w} =$ `abaab`:

i	1	2	3	4	5
$\boldsymbol{w}[i]$	a	b	a	a	b
$pr(\boldsymbol{w}[i])$	2	3	2	2	3
$Pr[i]$	2	6	12	24	72

$P\text{-}signature(\boldsymbol{w}[3\,..\,5]) =$
$P\text{-}signature(\texttt{aab}) =$
$Pr[5]/Pr[2] = 72/6 = 12$

i	1	2	3	4	5
$\boldsymbol{w}[i]$	a	b	a	a	b
$s(i)$	0	1	0	0	1
$S[i]$	0	1	1	1	2

$S\text{-}signature(\boldsymbol{w}[3\,..\,5]) =$
$S\text{-}signature(\texttt{aab}) =$
$S[5] - S[2] = 2 - 1 = 1$

3 Brute Force Algorithms

We will first focus on the case where we consider periods without head nor tail.

In the remaining of the article we will write that a word $\boldsymbol{w}$ has Abelian period p whenever it has Abelian period $(0, p)$. When the tail is also empty, for a word $\boldsymbol{w}$ of length n an Abelian period p must divide n. We define:

- $P[i]$ is the set of Abelian periods of $\boldsymbol{w}[1\,..\,i]$;
- $V[i] = \mathcal{P}(\boldsymbol{w}[1\,..\,i])$ is the Parikh vector of $\boldsymbol{w}[1\,..\,i]$.

3.1 Abelian periods with neither head nor tail

In a first step we set $P[i] = \{i\}$ for all the divisors of n. Then we process the positions i of $\boldsymbol{w}$ in ascending order: if $j \in P[i]$ and $\mathcal{P}_{\boldsymbol{w}}[i+1\,..\,i+j] = \mathcal{P}_{\boldsymbol{w}}[1\,..\,j]$, then we add j to $P[i+j]$. This test can be done in $O(\sigma)$ time by precomputing the Parikh vectors of all the prefixes of $\boldsymbol{w}$ or in constant time using signatures. At the end of the process $P[n]$ contains all the Abelian periods of $\boldsymbol{w}$ with neither head nor tail (see algorithm in Figure 1).

```
AbelianPeriodsNoHeadNoTail(w, n)
 1  V[i] ← P(w[1..i]), ∀ 1 ≤ i ≤ n
 2  P[i] ← ∅, ∀ 1 ≤ i ≤ n
 3  for i ← 1 to n/2 do
 4     if n mod i = 0 then
 5        P[i] ← {i}
 6  for i ← 1 to n − 1 do
 7     for j ∈ P[i] do
 8        if V[i + j] − V[i] = V[j] then
 9           P[i + j] ← P[i + j] ∪ {j}
10  return P[n]
```

Figure 1. Compute the Abelian periods with no head and no tail of a word $\boldsymbol{w}$ of length n

```
AbelianPeriodsNoHeadWithTail(w, n)
 1  V[i] ← P(w[1..i]), ∀ 1 ⩽ i ⩽ n
 2  P[i] ← {i}, ∀ 1 ⩽ i ⩽ n/2
 3  P[i] ← ∅, ∀ n/2 < i ⩽ n
 4  for i ← 1 to n − 1 do
 5     for j ∈ P[i] do
 6        if i + j > n then
 7           if V[n] − V[i + 1] ≤ V[j] then
 8              P[n] ← P[n] ∪ {j}
 9        else if V[i + j] − V[i] = V[j] then
10              P[i + j] ← P[i + j] ∪ {j}
11  return P[n]
```

Figure 2. Compute the Abelian periods without head and with a possibly non-empty tail of a word $\boldsymbol{w}$ of length n

Example 7. $\boldsymbol{w}$ = abaababbbabaabbabbaaabbababbaa:

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
$\boldsymbol{w}[i]$	a	b	a	a	b	a	b	b	b	a	b	a	a	b	b	a	b	b	a	a	a	b	b	a	b	a	b	b	a	a
P	{1}	{2}	{3}		{5}	{6}				{10}					{15}					{10}										{10}
						{3}																								

Theorem 8. *The algorithm* ***AbelianPeriodsNoHeadNoTail*** *computes all the Abelian periods with neither head nor tail of a word* $\boldsymbol{w}$ *of length* n *in time* $O(n^2 \times \sigma)$ *if the test in line 8 is performed by comparing Parikh vectors and in time* $O(n^2)$ *if the test in line 8 is performed by using S-signatures or P-signatures.*

3.2 Abelian periods without head with tail

Now we consider Abelian periods without head and with a possibly non-empty tail. We adapt the previous algorithm by setting $P[i] = \{i\}$ for $1 \leqslant i \leqslant n/2$ (see algorithm Figure 2).

Theorem 9. *The algorithm* ***AbelianPeriodsNoHeadWithTail*** *computes all the Abelian periods without head and with tail of a word* $\boldsymbol{w}$ *of length* n *in time* $O(n^2 \times \sigma)$ *if the tests in lines 7 and 1 are performed by comparing Parikh vectors and in time* $O(n^2)$ *if the test in lines 7 and 1 are performed by using S-signatures or P-signatures.*

4 Quasi-Linear Time Computation of Abelian Periods with neither Head nor Tail

In a linear-time preprocessing phase we compute $\mathcal{P}_{\boldsymbol{w}}[j]$, $j = 1, 2, \ldots, \sigma$, the components of the Parikh vector of the word $\boldsymbol{w}$. Also we compute

$$g = \gcd(\mathcal{P}_{\boldsymbol{w}}[1], \mathcal{P}_{\boldsymbol{w}}[2], \ldots, \mathcal{P}_{\boldsymbol{w}}[\sigma])$$

and $q = n/g$. Without loss of generality we suppose $\sigma \geq 2$ and $g > 1$. In $O(\sqrt{g})$ time we compute a stack D of all divisors $1 \leq d \leq g$ of g in ascending order.

Definition 10. *The word $\boldsymbol{w}$ is an **Abelian repetition** of **period** p and **exponent** e if $p \mid n$ and each of the e substrings*

$$\boldsymbol{w}[1\,.\,.\,p], \boldsymbol{w}[p+1\,.\,.\,2p], \ldots, \boldsymbol{w}[n-p+1\,.\,.\,n]$$

contains $(p \times \mathcal{P}_{\boldsymbol{w}}[j])/n = \mathcal{P}_{\boldsymbol{w}}[j]/e$ occurrences of the letter $\sigma_j \in \Sigma$ for any j.

In other words, an Abelian repetition of period p and exponent e is the concatenation of e strings all having the same Parikh vector $\mathcal{P}$ of length p.

Observation 3 *The only possible Abelian periods p of $\boldsymbol{w}$ are of the form $p = d \times q$, where d is an entry in D. Thus the smallest period is $d \times q$, where d is the least such entry.(Note that the last element of D is g.)*

Definition 11 (Segment). *A factor $\boldsymbol{w}[i\,.\,.\,j]$ is a **segment** of $\boldsymbol{w}$ if:*

1. *$i = k \times q + 1$ with $k \geqslant 0$;*
2. *$j - i + 1 = t \times q$ with $t \geqslant 1$;*
3. *$\mathcal{P}_{\boldsymbol{w}[i..j]}[k]/(j-i+1) = \mathcal{P}_{\boldsymbol{w}}[k]/|\boldsymbol{w}|$ for every letter $\sigma_k \in \Sigma$;*
4. *there does not exist a $j' < j$ such that $j'-i+1 = t' \times q$ and $\mathcal{P}_{\boldsymbol{w}[i..j']}[k]/(j'-i+1) = \mathcal{P}_{\boldsymbol{w}}[k]/|\boldsymbol{w}|$ for every letter $\sigma_k \in \Sigma$.*

In other words segments:

- start at positions multiples of q plus one;
- are non-empty and of length multiple of q;
- have the same proportion of every letter as the whole word $\boldsymbol{w}$;
- are of minimal length.

Since we suppose that $\boldsymbol{w}$ has Abelian period $p \in 1\,.\,.\,n/2$, it follows that either $\boldsymbol{w}$ itself is a segment or else consists of a concatenation of segments. Note that a segment is a minimum-length substring of Abelian period p.

Lemma 12. *The word $\boldsymbol{w}$ has Abelian period $d \times q$ if and only if for every $k = 0, 1, \ldots, n/(d \times q) - 1$, $k \times d \times q + 1$ is the starting position of a segment of $\boldsymbol{w}$.*

We begin by computing the segments of $\boldsymbol{w}$ (see Figure 3), making use of the precomputed values q and $\mathcal{P}_{\boldsymbol{w}}$. We compute a Boolean array L of n elements: for $1 \leqslant i \leqslant n$, $L[i] = 1$ iff i is the starting position of a segment, $L[i] = 0$ otherwise.

Observation 4 *If p is an Abelian period of $\boldsymbol{w}$ with neither head nor tail and T is the length of the longest segment of $\boldsymbol{w}$ divided by q, then $p \geqslant T$.*

```
ComputesSegments(w, n, q, P_w)
(i, T) ← (1, 0)
L ← 0^n
while i ⩽ n do
   ▷ Start a new segment
   (i_0, j, t, count) ← (i, 0, 0, 0^σ)
   while j ⩽ σ do
      ▷ See if t partitions of length q form a segment
      t ← t + 1
      for k ← 1 to q do
         j ← w[i]
         count[j] ← count[j] + 1
         i ← i + 1
      ▷ Check counts of letters 1 . . j from position i_0
      j ← 1
      t′ ← t × q
      while j ⩽ σ and count[j] = (t′ × P_w[j])/n do
         j ← j + 1
   ▷ Update the array L and the maximum segment length T
   L[i_0] ← 1
   T ← max{T, t}
return (L, T)
```

Figure 3. Compute a Boolean array L of the starting positions of the segments of $\boldsymbol{w}$ ordered from left to right, also the maximum number T of factors of length q in any segment

The procedure that computes L visits each position i in $\boldsymbol{w}$ once, and corresponding to each i performs constant-time processing: the internal **while** loop updates j at most σ times corresponding to each partition of length $q \geqslant \sigma$.

Proposition 13. *The algorithm* ComputesSegments$(\boldsymbol{w}, n, q, \mathcal{P}_{\boldsymbol{w}})$ *computes the segments of a word* $\boldsymbol{w}$ *of length* n *on an alphabet of size* σ *in time* $O(n)$.

Example 14. $\boldsymbol{w} = \texttt{abaababbbabaabbabbaaabbababbaa}$: $n = 30$, $\mathcal{P}_{\boldsymbol{w}} = (15, 15)$

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
$\boldsymbol{w}[i]$	a	b	a	a	b	a	b	b	b	a	b	a	a	b	b	a	b	b	a	a	a	b	b	a	b	a	b	b	a	a
$L[i]$	1	0	1	0	0	0	0	0	1	0	1	0	1	0	1	0	1	0	0	0	1	0	1	0	1	0	1	0	0	0
T	0	1	1	1	1	1	1	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3

$\boldsymbol{w}$ is thus a concatenation of segments: $\boldsymbol{w} = \texttt{ab} \cdot \texttt{aababb} \cdot \texttt{ba} \cdot \texttt{ba} \cdot \texttt{ab} \cdot \texttt{ba} \cdot \texttt{bbaa} \cdot \texttt{ab} \cdot \texttt{ba} \cdot \texttt{ba} \cdot \texttt{bbaa}$ and $T = 3$.

The procedure, given in Figure 4, scans all the multiples of the divisors $d \in D$, their number is equal to the sum of the divisors of g which is in $O(n \log\log n)$ [19].

In practice, the case where $d = 1$ is treated in lines 5 and 7. If $T = 1$, it means that w can be segmented into factors of length q: q is then an Abelian period of w. The case where $d = g$ is treated outside the main loop, at the end of the algorithm: it corresponds to the trivial case where the Abelian period is n.

Example 15. $\boldsymbol{w} = \texttt{abaababbbabaabbabbaaabbababbaa}$: $n = 30$, $\mathcal{P}_{\boldsymbol{w}}[1] = \mathcal{P}_{\boldsymbol{w}}[2] = 15$, $g = 15$, $q = 2$, $D = (1, 3, 5, 15)$ and $T = 3$. Since $T \neq 1$, q is not an Abelian period: case $d = 1$ is done. When $d = 3$, $p = 7$ and 7 is not a starting position of a segment. When $d = 5$, $p = 11$ and 11 is a starting position of a segment then $p = 21$

```
ComputesPeriod(w, n)
Compute P_w, g, D
q ← n/g
(L, T) ← ComputesSegments(w, n, q, P_w)
R ← ∅
▷ Deal quickly with easy cases
if T = 1 then
   R ← R ∪ {q}
   d ← Pop(D)
▷ Fast forward in D past impossible cases
repeat
   d ← Pop(D)
until d ⩾ T
while d < g do
   p ← d × q + 1
   ▷ Test if all multiples of p are starting positions of segments
   while p < n do
      if L[p] = 1 then
         p ← p + d × q
      else break
   if p ⩾ n then
      R ← R ∪ {d × q}
   d ← Pop(D)
if q ≠ n then
   R ← R ∪ {n}
return R
```

Figure 4. In ascending order of divisors d of g, use the array L to determine whether or not $\boldsymbol{w}$ is an Abelian repetition of period $d \times q$

and 21 is a starting position of a segment: 10 is an Abelian period. The case where $d = 15$ is trivial since it corresponds to Abelian period n. Thus the algorithm returns $\{10, 30\}$. In the worst case the algorithm could have scanned all the multiples of 3 (they are 5) and all the multiples of 5 (they are 3) less than or equal to 15.

Theorem 16. *The algorithm* ComputesPeriod$(\boldsymbol{w}, n)$ *computes all the Abelian periods of* $\boldsymbol{w}$ *in time* $O(n \log\log n)$.

5 Conclusions and perspectives

In this article we gave brute force algorithms for computing Abelian periods for a word $\boldsymbol{w}$ of length n in the two following cases: no head, no tail and no head with tail. These algorithms run in time $O(n^2)$ but is this complexity tight? We also present a quasi-linear time algorithm for computing all the Abelian periods of a word in the case no head, no tail. Does an algorithm of the same complexity exist for a word $\boldsymbol{w}$ of length at most $n + q - 1$ containing a substring of length n that is an Abelian repetition with neither head nor tail of some period $dq \leq n$?

References

1. S. Avgustinovich, A. Glen, B. Halldórsson, and S. Kitaev: *On shortest crucial words avoiding abelian powers.* Discrete Applied Mathematics, 158(6) 2010, pp. 605–607.
2. S. Avgustinovich, J. Karhumäki, and S. Puzynina: *On Abelian repetition threshold.* RAIRO Theoretical Informatics and Applications, 46(1) 2012, pp. 3–15.
3. F. Blanchet-Sadri, J. I. Kim, R. Mercas, W. Severa, and S. Simmons: *Abelian square-free partial words*, in Proceedings of the 4th International Conference Language and Automata Theory and Applications, A.-H. Dediu, H. Fernau, and C. Martín-Vide, eds., vol. 6031 of Lecture Notes in Computer Science, Springer, 2010, pp. 94–105.
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 Zs. Lipták: *On Table Arrangements, Scrabble Freaks, and Jumbled Pattern Matching*, in Proceedings of the 5th International Conference on Fun with Algorithms, FUN 2010, P. Boldi and L. Gargano, eds., vol. 6099 of Lecture Notes in Computer Science, Springer, 2010, pp. 89–101.
6. P. Burcsi, F. Cicalese, G. Fici, and Zs. Lipták: *Algorithms for jumbled pattern matching in strings.* International Journal of Foundations of Computer Science, 23(2) 2012, pp. 357–374.
7. P. Burcsi, F. Cicalese, G. Fici, and Zs. Lipták: *On Approximate Jumbled Pattern Matching in Strings.* Theory of Computing Systems, 50(1) 2012, pp. 35–51.
8. J. Cassaigne, G. Richomme, K. Saari, and L. Zamboni: *Avoiding Abelian powers in binary words with bounded Abelian complexity.* International Journal of Foundations of Computer Science, 22(4) 2011.
9. M. Christou, M. Crochemore, and C. S. Iliopoulos: *Identifying all Abelian periods of a string in quadratic time and relevant problems.* International Journal of Foundations of Computer Science, 2012, (accepted, see also Report arXiv:1207.1307v1).
10. F. Cicalese, G. Fici, and Zs. Lipták: *Searching for jumbled patterns in strings*, in Proceedings of the Prague Stringology Conference 2009, J. Holub and J. Žďárek, eds., Czech Technical University in Prague, Czech Republic, 2009, pp. 105–117.
11. S. Constantinescu and L. Ilie: *Fine and Wilf's theorem for abelian periods.* Bulletin of the European Association for Theoretical Computer Science, 89 2006, pp. 167–170.
12. L. J. Cummings and W. F. Smyth: *Weak repetitions in strings.* Journal of Combinatorial Mathematics and Combinatorial Computing, 24 1997, pp. 33–48.
13. J. D. Currie and A. Aberkane: *A cyclic binary morphism avoiding Abelian fourth powers.* Theoretical Computer Science, 410(1) 2009, pp. 44–52.
14. M. Domaratzki and N. Rampersad: *Abelian primitive words*, in Proceedings of the 15th Conference on Developments in Language Theory, G. Mauri and A. Leporati, eds., vol. 6795 of Lecture Notes in Computer Science, Springer, 2011.
15. G. Fici, T. Lecroq, A. Lefebvre, and É. Prieur-Gaston: *Computing Abelian periods in words*, in Proceedings of the Prague Stringology Conference 2011, J. Holub and J. Žďárek, eds., Czech Technical University in Prague, Czech Republic, 2011, pp. 184–196.
16. M. Lothaire: *Algebraic Combinatorics on Words*, Cambridge University Press, 2002.
17. T. M. Moosa and M. S. Rahman: *Indexing permutations for binary strings.* Information Processing Letters, 110(18-19) 2010, pp. 795–798.
18. T. M. Moosa and M. S. Rahman: *Sub-quadratic time and linear space data structures for permutation matching in binary strings.* Journal of Discrete Algorithms, 10 2012, pp. 5–9.
19. G. Robin: *Grandes valeurs de la fonction somme des diviseurs et hypothèse de Riemann.* J. Math. Pures Appl. (9), 63(2) 1984, pp. 187–213.
20. A. Samsonov and A. Shur: *On Abelian repetition threshold.* RAIRO Theoretical Informatics and Applications, 46(1) 2012, pp. 147–163.