A Note on Efficient Computation of All Abelian Periods in a String

M. Crochemore[a,b], C. S. Iliopoulos[a,c], T. Kociumaka[d], M. Kubica[d], J. Pachocki[d], J. Radoszewski[d,*], W. Rytter[d,e,1], W. Tyczyński[d], T. Waleń[f,d]

[a] *King's College London, London WC2R 2LS, UK*
[b] *Université Paris-Est, France*
[c] *Digital Ecosystems & Business Intelligence Institute, Curtin University of Technology, Perth WA 6845, Australia*
[d] *Faculty of Mathematics, Informatics and Mechanics, University of Warsaw, ul. Banacha 2, 02-097 Warsaw, Poland*
[e] *Dept. of Math. and Informatics, Copernicus University, ul. Chopina 12/18, 87-100 Toruń, Poland*
[f] *Laboratory of Bioinformatics and Protein Engineering, International Institute of Molecular and Cell Biology in Warsaw, Poland*

Abstract

We derive a simple efficient algorithm for Abelian periods knowing all Abelian squares in a string. An efficient algorithm for the latter problem was given by Cummings and Smyth in 1997. By the way we show an alternative algorithm for Abelian squares. We also obtain a linear time algorithm finding all "long" Abelian periods. The aim of the paper is a (new) reduction of the problem of all Abelian periods to that of (already solved) all Abelian squares which provides new insight into both connected problems.

Keywords: algorithms, Abelian period, Abelian square

1. Introduction

We present an efficient reduction of the Abelian period problem to the Abelian square problem. For a string of length n the latter problem was solved in $O(n^2)$ by Cummings and Smyth [6]. The best previously known algorithms for the Abelian periods, see [11], worked in $O(n^2 m)$ time (where m is the alphabet size) which for large m is $O(n^3)$. Our algorithm works in $O(n^2)$ time, independently of the alphabet size. As a by-product we obtain an alternative

*Corresponding author. Tel.: +48-22-55-44-484, fax: +48-22-55-44-400.
Email addresses: maxime.crochemore@kcl.ac.uk (M. Crochemore), c.iliopoulos@kcl.ac.uk (C. S. Iliopoulos), kociumaka@mimuw.edu.pl (T. Kociumaka), kubica@mimuw.edu.pl (M. Kubica), pachocki@mimuw.edu.pl (J. Pachocki), jrad@mimuw.edu.pl (J. Radoszewski), rytter@mimuw.edu.pl (W. Rytter), w.tyczynski@mimuw.edu.pl (W. Tyczyński), walen@mimuw.edu.pl (T. Waleń)
[1] The author is supported by grant no. N206 566740 of the National Science Centre.

$O(n^2)$ time algorithm finding all Abelian squares and an $O(n)$ time algorithm finding a compact representation of all Abelian periods of length greater than $n/2$, in particular, the shortest such period.

Abelian squares were first studied by Erdös [10], who posed a question on the smallest alphabet size for which there exists an infinite Abelian-square-free string. An example of such a string over five-letter alphabet was given by Pleasants [15] and afterwards the best possible example over four-letter alphabet was shown by Keränen [12].

Quite recently there have been several results on Abelian complexity in words [1, 4, 7, 8, 9] and partial words [2, 3] and on Abelian pattern matching [13, 14]. Abelian periods were first defined and studied by Constantinescu and Ilie [5].

We say that two strings are (commutatively) equivalent, and write $x \equiv y$, if one can be obtained from the other by permuting its symbols. In other words, the Parikh vectors $\mathcal{P}(x)$, $\mathcal{P}(y)$ are equal, where the Parikh vector gives frequency of each symbol of the alphabet in a given string. Parikh vectors were introduced already in [5] for this problem.

A string w is an *Abelian k-power* if $w = x_1 x_2 \dots x_k$, where

$$x_1 \equiv x_2 \equiv \dots \equiv x_k$$

The size of x_1 is called the *base* of the k-power. In particular w is an Abelian square if and only if it is an Abelian 2-power.

A string x is an Abelian factor of y if $\mathcal{P}(x) \leq \mathcal{P}(y)$, that is, each element of $\mathcal{P}(x)$ is smaller than the corresponding element of $\mathcal{P}(y)$. The pair (i, p) is an *Abelian period* of $w = w[1, n]$ if and only if $w[i+1, j]$ is an Abelian k-power with base p (for some k) and $w[1, i]$ and $w[j+1, n]$ are Abelian factors of $w[i+1, i+p]$, see Fig. 1. Here p is called the *length* of the period.

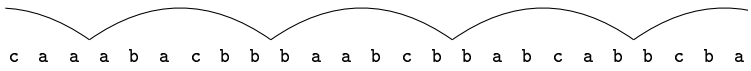

Figure 1: A word of length 25 with an Abelian period $(i = 3, p = 6)$. This period implies two Abelian squares: `abacbbbaabcb` and `baabcbbabcab`.

In Section 2 we introduce two auxiliary tables that we use in computing Abelian squares and powers. Next in Section 3 we show new $O(n^2)$ time algorithms for all Abelian squares and all Abelian periods in a string and a reduction between these problems. Finally in Section 4 we present an $O(n)$ time algorithm finding a compact representation of all "long" Abelian periods. The representation has the form (i_1, a_1, b_1), (i_2, a_2, b_2),... where (i_j, a_j, b_j) denotes all Abelian periods of the form (i_j, p) for $p \in [a_j, b_j]$.

2. Auxiliary tables

Let w be a string of length n. Assume its positions are numbered from 1 to n, $w = w_1 w_2 \dots w_n$. By $w[i, j]$ we denote the factor of w of the form $w_i w_{i+1} \dots w_j$.

Factors of the form $w[1,i]$ are called prefixes of w and factors of the form $w[i,n]$ are called suffixes of w.

We introduce the following table:

$$head(i,j) \;=\; \text{minimum } k \text{ such that } \mathcal{P}(w[i,j]) \leq \mathcal{P}(w[j+1,k]).$$

If no such k exists, we set $head(i,j) = \infty$.

Example 1. *For the infinite Fibonacci word $\mathcal{F} = abaababaabaababaababa\ldots$ the first several values of the table $head(1,i)$ are:*

i	*1*	*2*	**3**	*4*	**5**	**6**	*7*	**8**	*9*	**10**	**11**	...
$\mathcal{F}[i]$	*a*	*b*	*a*	*a*	*b*	*a*	*b*	*a*	*a*	*b*	*a*	...
$head(1,i)$	*3*	*5*	**6**	*9*	**10**	**12**	*15*	**16**	*19*	**20**	**22**	...

We have here Abelian square prefixes of lengths 6, 10, 12, 16, 20, 22.

We show how to compute the *head* table in $O(n^2)$ time. The computation is performed in row-order of the table using the fact that it is non-decreasing:

Observation 2. *For any $1 \leq i \leq j < n$, $head(i,j) \leq head(i,j+1)$.*

We assume that the alphabet of w is $\Sigma = \{1,2,\ldots,m\}$ where $m \leq n$. For a Parikh vector Q, by $Q[i]$ for $i = 1,2,\ldots,m$ we denote the number of occurrences of the letter i. For two Parikh vectors Q and R, we define their *Parikh difference*, denoted as $Q - R$, as a Parikh vector: $(Q-R)[i] = Q[i] - R[i]$.

In the algorithm we store the difference $\Delta_j = \mathcal{P}(y_j) - \mathcal{P}(x_j)$ of Parikh vectors of

$$x_j = w[i,j] \quad \text{and} \quad y_j = w[j+1,k]$$

where $k = head(i,j)$. Note that $\Delta_j[a] \geq 0$ for any $a = 1,2,\ldots,m$.

Assume we have computed $head(i,j-1)$ and Δ_{j-1}. When we proceed to j, we move the letter $w[j]$ from y to x and update Δ accordingly. Thus at most one element of Δ might have dropped below 0. If there is no such element, we conclude that $head(i,j) = head(i,j-1)$ and that we have obtained $\Delta_j = \Delta$. Otherwise we keep extending y to the right with new letters and updating Δ until all its elements become non-negative. We obtain the following algorithm Compute-*head*.

Lemma 3. *The head table can be computed in $O(n^2)$ time.*

Proof. The time complexity of the algorithm Compute-*head* is $O(n^2)$. Indeed, the total number of steps of the while-loop for a fixed value of i is $O(n)$, since each step increases the variable k. □

We also use the following *tail* table which is, in fact, the *head* table for a reversed word:

$$tail(i,j) \;=\; \text{maximum } k \text{ such that } \mathcal{P}(w[i-j+1,i]) \leq \mathcal{P}(w[k,i-j]).$$

```
Algorithm Compute-head(w)
    for i := 1 to n do
        Δ := (0, 0, ..., 0);
        Δ[w[i]] := 1; {Boundary condition}
        k := i;
        for j := i to n do
            Δ[w[j]] := Δ[w[j]] − 2;
            while (k < n) and (Δ[w[j]] < 0) do
                k := k + 1;
                Δ[w[k]] := Δ[w[k]] + 1;
            if Δ[w[j]] < 0 then k := ∞;
            head(i, j) := k;
```

3. Abelian squares and Abelian periods

In this section we show how Abelian periods can be inferred from Abelian squares in a string.

Define by $maxpower(i,p)$ the maximal size of a prefix of $w[i,n]$ which is an Abelian k-power with base p (for some k). Define $square(i,p) = 1$ if and only if $maxpower(i,p) \geq 2p$. Cummings and Smyth [6] compute an alternative table $square'(i,p)$, such that $square'(i,p) = 1$ if and only if $w[i-p+1, i+p]$ is an Abelian square. These tables are clearly equivalent:

$$square(i,p) = 1 \;\Leftrightarrow\; square'(i+p-1,p) = 1.$$

The $maxpower(i,p)$ table can be computed from the $square(i,p)$ table in linear time using a simple dynamic programming recurrence:

$$maxpower(i,p) = \begin{cases} 0 & \text{if } n-i < p-1 \\ p + square(i,p) \cdot maxpower(i+p,p) & \text{otherwise.} \end{cases} \tag{1}$$

An alternative $O(n^2)$ time algorithm for computing the table $square(i,p)$ for a string w of length n is a consequence of the following observation, see also Example 1.

Observation 4. $square(i,p) = 1 \;\Leftrightarrow\; head(i, i+p-1) = i+2p-1$.

Theorem 5. *All Abelian squares in a string of length n can be computed in $O(n^2)$ time.*

The following observation provides a constant-time condition for checking an Abelian period.

Observation 6. *(i,p) is an Abelian period of w if and only if*

$$p \geq head(1,i), tail(n, n-j+1)$$

where $j = i + 1 + maxpower(i+1, p)$.

We conclude with the following algorithm for computing Abelian periods. In the algorithm we use our alternative version of computing the table *square* from *head*, since the latter table is computed anyway (instead of that Cummings and Smyth's algorithm can be used for Abelian squares).

```
Algorithm Compute-Abelian-Periods
    Compute head(i,j), tail(i,j) using algorithm Compute-head;
    Initialize the table maxpower to zero table;
    for p := 1 to n do
      for i := n downto 1 do
        if i ≤ n − p + 1 then
          maxpower(i,p) := p;
          if head(i, i+p−1) = i + 2p − 1 then
            maxpower(i,p) := p + maxpower(i+p, p);
    for i := 1 to n do
      for p := 1 to n do
        j = i + 1 + maxpower(i+1, p);
        if (p ≥ head(1,i)) and (p ≥ tail(n, n−j+1)) then
          Report an Abelian period (i,p);
```

Theorem 7. *All Abelian periods of a string of length n can be computed in $O(n^2)$ time.*

4. Long Abelian periods

Consider an Abelian period (i,p) of w such that $p > n/2$, we call it a *long* Abelian period. We have $w = u_0u_1u_2$ with $|u_0| = i$, $|u_1| = p$, $|u_2| = j$ and $\mathcal{P}(u_0), \mathcal{P}(u_2) \leq \mathcal{P}(u_1)$. Equivalently,

$$n - i - j = p \geq head(1,i), tail(n,j),$$

which can be reformulated as follows:

Observation 8. *(i,p) is a long Abelian period of w if and only if*

$$j \geq 0 \quad \wedge \quad j \leq n - i - head(1,i) \quad \wedge \quad tail(n,j) + j \leq n - i \quad \wedge \quad j \leq \left\lceil \frac{n}{2} \right\rceil - i - 1$$

where $j = n - i - p$.

We use the condition from Observation 8 to iterate through all possible lengths of the head $i = 0, 1, \ldots$ and find the interval of possible values of the tail (i.e. j). Note that the function $tail(n, j) + j$, for $j = 0, 1, \ldots, n$, is increasing (since it is a sum of a non-decreasing function and an increasing function). Let $W(k)$, for $0 \leq k \leq n$, be:

$$W(k) = \max\{j \,:\, tail(n, j) + j \leq k\}.$$

If no such j exists, we set $W(k) = -\infty$. Note that this table can be computed in $O(n)$ time.

Using the table $W(k)$, for a given head i we can state a simple characterization of tails of all possible long periods with the head i:

$$0 \leq j \leq \min(n - i - head(1, i), W(n - i), \left\lceil \frac{n}{2} \right\rceil - i - 1).$$

We obtain the following $O(n)$ time algorithm for computing a compact representation of all the long periods and, in particular, the shortest such period.

Algorithm Compute-Long-Abelian-Periods(w)
  Compute the tables $head(1, i)$, $tail(n, i)$
  { assume $head(1, 0) = tail(n, 0) = 0$ }
  for $k := 0$ **to** n **do** $W(k) := -\infty$
  for $j := 0$ **to** $\lceil n/2 \rceil - 1$ **do**
    if $j + tail(n, j) \leq n$ **then**
      $W(tail(n, j) + j) := j$
  for $k := 1$ **to** n **do**
    $W(k) := \max(W(k), W(k - 1))$
  $shortest := n$
  for $i := 0$ **to** $\lceil n/2 \rceil - 1$ **do**
    $up := \min(n - i - head(1, i), W(n - i), \lceil n/2 \rceil - i - 1)$
    report the Abelian periods $\{(i, n - i - j) \,:\, 0 \leq j \leq up\}$
    if $up \geq 0$ **then**
      $shortest := \min(shortest, n - i - up)$
  return *shortest*

Theorem 9. *For a string of length n, a compact representation of all Abelian periods of length greater than $n/2$ can be computed in $O(n)$ time.*

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] 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, *LATA*, volume 6031 of *Lecture Notes in Computer Science*, pages 94–105. Springer, 2010.

[3] F. Blanchet-Sadri and S. Simmons. Avoiding Abelian powers in partial words. In Mauri and Leporati [13], pages 70–81.

[4] J. Cassaigne, G. Richomme, K. Saari, and L. Q. Zamboni. Avoiding Abelian powers in binary words with bounded Abelian complexity. *Int. J. Found. Comput. Sci.*, 22(4):905–920, 2011.

[5] S. Constantinescu and L. Ilie. Fine and Wilf's theorem for Abelian periods. *Bulletin of the EATCS*, 89:167–170, 2006.

[6] L. J. Cummings and W. F. Smyth. Weak repetitions in strings. *J. Combinatorial Math. and Combinatorial Computing*, 24:33–48, 1997.

[7] J. D. Currie and A. Aberkane. A cyclic binary morphism avoiding Abelian fourth powers. *Theor. Comput. Sci.*, 410(1):44–52, 2009.

[8] J. D. Currie and T. I. Visentin. Long binary patterns are Abelian 2-avoidable. *Theor. Comput. Sci.*, 409(3):432–437, 2008.

[9] M. Domaratzki and N. Rampersad. Abelian primitive words. In Mauri and Leporati [13], pages 204–215.

[10] P. Erdös. Some unsolved problems. *Hungarian Academy of Sciences Mat. Kutató Intézet Közl.*, 6:221–254, 1961.

[11] G. Fici, T. Lecroq, A. Lefebvre, and É. Prieur-Gaston. Computing Abelian periods in words. In J. Holub and J. Žďárek, editors, *Proceedings of the Prague Stringology Conference 2011*, pages 184–196, Czech Technical University in Prague, Czech Republic, 2011.

[12] V. Keränen. Abelian squares are avoidable on 4 letters. In W. Kuich, editor, *ICALP*, volume 623 of *Lecture Notes in Computer Science*, pages 41–52. Springer, 1992.

[13] G. Mauri and A. Leporati, editors. *Developments in Language Theory - 15th International Conference, DLT 2011, Milan, Italy, July 19-22, 2011. Proceedings*, volume 6795 of *Lecture Notes in Computer Science*. Springer, 2011.

[14] T. M. Moosa and M. S. Rahman. Indexing permutations for binary strings. *Inf. Process. Lett.*, 110(18-19):795–798, 2010.

[15] P. A. Pleasants. Non-repetitive sequences. *Proc. Cambridge Phil. Soc.*, 68:267–274, 1970.