Information Processing Letters 113 (2013) 74–77

Contents lists available at SciVerse ScienceDirect

Information Processing Letters

www.elsevier.com/locate/ipl

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] Department of Mathematics 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

ARTICLE INFO

Article history:
Received 27 July 2012
Received in revised form 31 October 2012
Accepted 2 November 2012
Available online 5 November 2012
Communicated by M. Yamashita

Keywords:
Algorithms
Abelian period
Abelian square

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.

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 [7]. The best previously known algorithms for the Abelian periods, see [12], worked in $O(n^2m)$ time (where m is the alphabet size) which for large m is $O(n^3)$. Our algorithm works in $O(n^2)$ time. As a by-product we obtain an alternative $O(n^2)$ 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 [11], 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 [16] and afterwards the best possible example over four-letter alphabet was shown by Keränen [13].

Quite recently there have been several results on Abelian complexity in words [1,4,8–10] and partial words [2,3] and on Abelian pattern matching [5,14,15]. Abelian periods were first defined and studied by Constantinescu and Ilie [6].

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 [6] for this problem.

* Corresponding author. Tel.: +48 22 55 44 484; fax: +48 22 55 44 400.
E-mail 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.

0020-0190/$ – see front matter
http://dx.doi.org/10.1016/j.ipl.2012.11.001

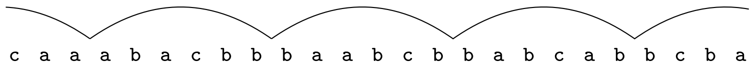

Fig. 1. A word of length 25 with an Abelian period ($i = 3$, $p = 6$). This period implies two Abelian squares: abacbbbaabcb and baabcbbabcab.

Table 1
The values of $head(1, i)$, $i = 1, \ldots, 11$, for the infinite Fibonacci word. Numbers in bold denote halves of square prefixes of the word.

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*	...
$head(1, i)$	2	3	**3**	5	**5**	**6**	8	**8**	10	**10**	**11**	...

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

$$x_1 \equiv x_2 \equiv \cdots \equiv x_k.$$

The length 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 $\mathcal{P}(x) \leqslant \mathcal{P}(y)$, 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.

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. Define

$$MinLong(i) = \min\{p > n/2\colon\ (i, p) \text{ is an Abelian period of } w\}.$$

If no such p exists, we set $MinLong(i) = \infty$. All long Abelian periods are of the form (i, p) where $p \geqslant MinLong(i)$, the table *MinLong* is a *compact* $O(n)$ space representation of potentially quadratic set of long Abelian periods.

2. Auxiliary tables

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

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

If no such k exists, we set $head(i, j) = \infty$, and if $j < i$, we set $head(i, j) = 0$. In the algorithm below we actually compute a slightly modified table $head'(i, j) = j + head(i, j)$.

Example 1. For the infinite Fibonacci word $\mathcal{F} = abaababaabaababaababaa\ldots$ the first several values of the table $head(1, i)$ are presented in Table 1.

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 $1 \leqslant i \leqslant j < n$, $head'(i, j) \leqslant head'(i, j+1)$.

We assume that the alphabet of w is $\Sigma = \{1, 2, \ldots, m\}$ where $m \leqslant n$. 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 $\Delta_j[a] \geqslant 0$ 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 that is analogical to the *head* table:

$$tail(i, j) = \text{minimum } k \text{ such that } \mathcal{P}\big(w[i, j]\big) \leqslant \mathcal{P}\big(w[i-k, i-1]\big).$$