# Computing Abelian Covers and Abelian Runs 

Shohei Matsuda, Shunsuke Inenaga, Hideo Bannai, and Masayuki Takeda<br>Department of Informatics, Kyushu University, Japan<br>\{shohei.matsuda, inenaga, bannai, takeda\}@inf.kyushu-u.ac.jp


#### Abstract

Two strings $u$ and $v$ are said to be Abelian equivalent if $u$ is a permutation of the characters of $v$. We introduce two new regularities on strings w.r.t. Abelian equivalence, called Abelian covers and Abelian runs, which are generalizations of covers and runs of strings, respectively. We show how to determine in $O(n)$ time whether or not a given string $w$ of length $n$ has an Abelian cover. Also, we show how to compute an $O\left(n^{2}\right)$-size representation of (possibly exponentially many) Abelian covers of $w$ in $O\left(n^{2}\right)$ time. Moreover, we present how to compute all Abelian runs in $w$ in $O\left(n^{2}\right)$ time, and state that the maximum number of all Abelian runs in a string of length $n$ is $\Omega\left(n^{2}\right)$.


Keywords: Abelian equivalence on strings, Parikh vectors, Abelian repetitions, covers of strings, string algorithms

## 1 Introducti

The study of Abelr seen in the paper if $u$ is a permutati and baaba are Abe attention and has
ys dates back to at least the early 60 's, as ngs $u, v$ are said to be Abelian equivalent pearing in $v$. For instance, strings aabba equivalence of strings has attracted much in several contexts.
blem called the jumbled pattern matching problem is to determine whether there is a substring of an input text string $w$ that is Abelian equivalent to a given pattern string $p$. There is a folklore algorithm to solve this problem in $O(n+m+\sigma)$ time using $O(\sigma)$ space, where $n$ is the length of $w, m$ is the length of $p$, and $\sigma$ is the alphabet size. Assuming $m \leq n$ and all characters appear in $w$, the algorithm runs in $O(n)$ time and $O(\sigma)$ space. The indexed version of the jumbled pattern matching proble where string belian


Shohei Matsuda, Shunsuke Inenaga, Hideo Bannai, Masayuki Takeda: Computing Abelian Covers and Abelian Runs, pp. 43-51.
Proceedings of PSC 2014, Jan Holub and Jan Ždárek (Eds.), ISBN 978-80-01-05547-2 © Czech Technical University in Prague, Czech Republic
where $y$ has some full Abelian period $d$, and $z$ is a non-empty string shorter than $d$ such that the number of each character $a$ contained in $z$ is no mo contained in the prefix $y[1 . . d]$ of length $d$ of $y$. A string $w$ is said to Abelian period $d$ if $w=x y$, where $y$ has some Abelian period $d$, and empty string sho e number of each character $a$ $x$ is no more thar orefix $y[1 . . d]$ of length $d$ of $y$. proposed an $O(n$ $O\left(n^{2}\right)$-time algori a to compute all lian periods for 1 optimal $O(n)$-t
contains arbitrary construction alphabet of s

Cummings a $O\left(n^{2}\right)$ time.
at
rd
d,
ains no
construc
algorithm
a given s
compute all Abelian squares in
${ }^{2}$ ) Abelian squares
nother $O\left(n^{2}\right)$-time algorithm
egularities on strings w.r.t. A
uns, which are generalizations
we propose non-trivial algor
cegularities. A set $C$ of intervals is called an
ostrings corresponding to the intervals in $C$ a
osition in $w$ is contained in at least one inte
$\mathrm{g} w$ of length $n$, we can determine whether or not $w$ has an

Abelian cover ir compute an $O(r$ covers of $w$. A s substring which by Cummings az runs in a string compute all Abe
ilso, we present an $O\left(n^{2}\right)$-time algorithm to of all (possibly exponentially many) Abelian to be an Abelian run of $w$ if $s$ is a maximal ciod. As a direct consequence from the result vn that the maximum number of all Abelian hen, we propose an $O\left(n^{2}\right)$-time algorithm to ing of length $n$.

## 2 Preliminartes

Let $\Sigma=\left\{c_{1}, \ldots, c_{\sigma}\right\}$ be an ordered alphabet. We assume that for each $c_{i} \in \Sigma$, its rank $i$ in $\Sigma$ is already known and can be computed in constant time. An element of $\Sigma^{*}$ is called a string. The length of a string $w$ is denoted by $|w|$. The empty string $\varepsilon$ is the string of length 0 , namely, $|\varepsilon|=0$. For a string $w=x y z$, strings $x, y$, and $z$ are called a prefix, substring, and suffix of $w$, respectively. The $i$-th character of a string $w$ of length $n$ is denoted by $w[i]$ for $1 \leq i \leq n$. For $1 \leq i \leq j \leq n$, let $w[i . . j]=w[i] \cdots w[j]$,


Figure 1. String aabbaabababa over a binary alphabet $\Sigma=\{\mathrm{a}, \mathrm{b}\}$ has an Abelian cover $\{[1,3],[4,6],[6,8],[8,10],[10,12]\}$ of length 3 with Parikh vector $\langle 2,1\rangle$, an Abelian cover $\{[1,4],[4,7],[7,10],[9,12]\}$ of length 4 with Parikh vector $\langle 2,2\rangle$, an Abelian cover $\{[1,5],[4,8],[8,12]\}$ of length 5 with Parikh vector $\langle 3,2\rangle$, and an Abelian cover $\{[1,11],[2,12]\}$ of length 11 with Parikh vector $\langle 6,5\rangle$. We remark that this string has other Abelian covers than the above ones.
i.e., $w[i . . j]$ is the substring of $w$ starting at position $i$ and ending at position $j$ in $w$. For convenience, let $w[i . . j]=\varepsilon$ if $j<i$. For any $0 \leq i \leq n$, strings $w[1 . . i]$ and $w[i . . n]$ are called prefixes and suffixes of $w$, respectively.

For any string $w$ of length $n \geq 2$, a set $I=\left\{\left[b_{1}, e_{1}\right], \ldots,\left[b_{|I|}, e_{|I|}\right]\right\}$ of intervals is called a cover of $w$ if $\left.\bigcup_{1 \leq k}, e_{k}\right] \neq[1, n]$ for every $1 \leq k \leq|I|$. Whenever we write $C=\{$ a cover $C$ of a string $w$, then we assume that $b_{j}<b_{j+1}$ for

Two strings $v, w \in \Sigma^{*}$ the characters in $w$. A Par of length $\sigma$ such that for character $c_{i}$ in $w$. Let $\preceq$ be $v, w \in \Sigma^{*}$ such that

$$
\begin{array}{ll}
P_{v}=P_{w} & \text { if } P_{v}[i]=P_{w}[i] \text { for all } 1 \leq i \leq \sigma, \text { and } \\
P_{v} \prec P_{w} & \text { if } P_{v} \neq P_{w} \text { and } P_{v}[i] \leq P_{w}[i] \text { for all } 1 \leq i \leq \sigma .
\end{array}
$$

For instance, for strings $v=$ aababc and $w=$ baba over an ordered alphabet $\Sigma=$ $\{\mathrm{a}, \mathrm{b}, \mathrm{c}\}, P_{v}=[3,2,1]$ and $P_{w}=[2,2,0]$, and therefore $P_{w} \prec P_{v}$. Clearly, strings $v, w$ are Abelian equivalent iff $P_{v}=P_{w}$. For any two strings $v, w \in \Sigma^{*}$, let $P_{v} \oplus P_{w}=P_{v w}$, namely, $\left(P_{v} \oplus P_{w}\right)[i]=P_{v}[i]+P_{w}[i]$ for each 1

quivalent if $v$ is a permutation of $\xi w \in \Sigma^{*}$, denoted $P_{w}$, is an array res the number of occurrences of vectors $P_{v}$ and $P_{w}$ for any strings

A cover $C=\left\{\left[b_{1}, e_{1}\right], \ldots,\left[b_{|C|}, e_{|C|}\right]\right\}$ of a $s$ $w$ if $P_{w\left[b_{1} . . e_{1}\right]}=P_{w\left[b_{j} . . e_{j}\right]}$ for all $1<j \leq|C|$. C $1<j \leq|C|$. The length and size of an Abelian a string are $e_{1}-b_{1}+1$ and $|C|$, respectively. covers of a string.

A non-empty substring $u$ of a string $w$ is call f $|u|$ is a multiple of an integer $d\left(1 \leq d \leq \frac{|u|}{2}\right)$ all $1 \leq k<\frac{|u|}{d}$. If $d=\frac{|u|}{2}$, then $u$ is called an $A$ elian cover of 1 holds for all , $\left.\left[b_{|C|}, e_{|C|}\right]\right\}$ of les of Abelian with period $d$ $k d+1 . .(k+1) d]$ for bstring $w[i . . j]$ of a string $w$ is called a maximal Abelian repetition of $w$ if $w[i . . j]$ is a non-extensible Abelian repetition with period $d$ in $w$, namely, if $w[i . . j]$ is an Abelian repetition satisfying (1) $P_{w[i-d . . i-1]} \neq P_{w[i . i+d-1]}$ or $i-d<0$ and (2) $P_{w[j-d+1 . . j]} \neq P_{w[j+1 . . j+d]}$ or $j+d>n$. A substring $w\left[i-h . . j+h^{\prime}\right]$ of a string $w$ is called an Abelian run


Figure 2. String aabbacbbaabbbb over a ternary alphabet $\Sigma=\{a, b, c\}$ has Abelian runs $(0,1,2,1,0),(0,3,4,1,0),(0,7,8,1,0),(0,9,10,1,0)$, and $(0,11,14,1,0)$ of period 1 , Abelian runs $(1,2,5,2,0),(1,8,11,2,1)$, and $(0,11,14,2,0)$ of period 2, and an Abelian run $(0,7,12,3,2)$ of period 3.
of $w$ if $w[i . . j]$ is a maximal $A$ period $d$ of $w$ and $h, h^{\prime} \geq 0$ are the largest integers satisfy respectively. Each Abelian run 5 -tuple ( $h, i, j, d, h^{\prime}$ ), where $h$ the Abelian run, respectively.

In this paper, we consider
Problem 1 (Abelian cover exis has an Abelian cover.

Problem 2 (All Abelian covers). Given a string $w$, compute all Abelian covers of $w$.
Problem 3 (All Abelian runs). Given a string $w$, compute all Abelian runs in $w$.

## 3 Algorithms

In this section, we present our algorithms to solve the problems stated in the previous section. For simplicity, we assume that all characters in $\Sigma$ appear in a given string $w$ of length $n$, which implies $\sigma \leq$

### 3.1 Abelian cover existen

In this subsection, we conside Abelian cover of a given strin sequence of strings over a binar has no Abelian covers), and th is a key to our solution to the

Lemma 4. If there exists an Abelia whether there exists an there exists an infinite oelian covers (e.g., $\mathrm{a}^{n-1} \mathrm{~b}$ st. The following lemma size for a string $w$, then there exists

$-d]$ and $P_{w\left[j+1 . . j+h^{\prime}\right]} \prec P_{w[j-d . j]}$, iod $d$ in $w$ is represented by a ft hand and the right hand of les of Abelian runs in a string.

## $v$, determine whether or not $w$




Clearly $w$ has an Abelian cover of size 2 iff $d_{j}=0$ for some $j$. For any $1 \leq j \leq\left\lfloor\frac{n}{2}\right\rfloor$, let $w[j]=c_{s}$ and $w[n-j+1]=c_{t}$. We can update $P_{w[1 . . j-1]}\left(\right.$ resp. $\left.P_{w[n-j . n]}\right)$ to $P_{w[1 . . j]}$ (resp. $\left.P_{w[n-j+1 . . n]}\right)$ in $O(1)$ time, increasing the value stored in the $s$ th entry (resp. the $t$ th entry) by 1 . Also, $d_{j}$ can be computed in $O(1)$ time from $d_{i-1}, P_{u, 11 i=11}[s]$, $P_{w[1 . . j]}[s], P_{w[n-j . n]}[t]$, and $P_{w[n-j+1 . . n]}[t]$. He time. The extra working space of the algorit we maintain.

The following corollary is immediate fro
Corollary 6. We can compute the longest $n$ in $O(n)$ time with $O(\sigma)$ wo

### 3.2 All Abelian covers

In this subsection, we conside string $w$ of length $n$. Note th exponentially large w.r.t. $n$. F of length at least $\left\lceil\frac{n}{2}\right\rceil$. This is $1, n]\}$ and any subset of $\{[2$,
ing all Abelian covers of a given belian covers of a string can be as $\sum_{k=\left\lceil\frac{n}{2}\right\rceil}^{n-1} 2^{n-k-1}$ Abelian covers $\left.\frac{n}{2}\right]$, the union of $\{[1, k],[n-k+$ $-k, n-1]\}$ is an Abelian cover of length $k$ for $a^{n}$. Therefore, we consider to compute a "compact" representation of all Abelian covers of a given string.

Theorem 7. Given a string $w$ of length $n$, we can compute an $O\left(n^{2}\right)$-size representation of all Abelian covers of $w$ in $O\left(n^{2}\right)$ time and $O(n)$ working space. Given a set $I$ of $s$ intervals sorted by the beginning positions of the intervals, the representation allows us to check if $I$ is an Abelian cover of $w$ in $O(s)$ time.

Proof. For each $1 \leq \ell \leq n-1$, we compute a subset $S_{\ell}$ of positions in $w$ such that $S_{\ell}=\left\{i \mid P_{w[i . i+\ell-1]}=P_{w[1 . . \ell]}, 1 \leq i \leq n-\ell+1\right\}$. Then, there exists an Abelian cover
of length $\ell$ for $w$ iff the distance between any two adjacent positions in $S_{\ell}$ is at most $\ell$. If $S_{\ell}$ satisfies the above condition, then we represent $S_{\ell}$ as a bit vector $B_{\ell}$ of length $n$ such that $B_{\ell}[i]=1$ if $i \in S_{\ell}$, and $B_{\ell}[i]=0$ otherwise. If $S_{\ell}$ does not satisfy the above
 d it. Now, given a set $I$ of $s$ intervals sorted by the beginning we first check if $I$ is a cover of $w$ and if each interval is of f $O(s)$ time. If $I$ satisfies both conditions, then we can check $P(s)$ time, using the bit vector $B_{\ell}$. Using a similar method $\leq \ell \leq n-1, S_{\ell}$ and its corresponding bit vector $B_{\ell}$ can ne. Hence, the overall time complexity of the algorithm is e (excluding the output) is $O(n)$, sin

Given a set $I$ of $s$ intervals, a naïve algorithm to check w cover of length $\ell$ requires $O(s \ell)$ time. Therefore, the solution o query time is more efficient th

### 3.3 All Abelian runs

In this subsection, we conside string $w$ of length $n$. We follo

ing all Abelian ru s by Cummings a epetitions in a st ower bound on $t l$
aababbaba $)^{n}$ of length $8 n$ has $\Theta\left(n^{2}\right)$ maximal Abelian repAbelian squares).
belian runs in a string is equal to that of maximal Abelian the following theorem is immediate:
Theorem 9. The maximum number of Abelian runs in a string $w$ of length $n$ is $\Omega\left(n^{2}\right)$.

Next, we show how to compute all Abelian runs in a given string.

| Theorem 11 <br> in $O\left(n^{2}\right)$ tim | length $n$, we can compute all Abelian runs in $w$ |
| :--- | :--- |
| Proof. We fir |  |
| Cummings a |  |
| that | $1 \leq i \leq n$, we compute a set $L_{i}$ of integers such |
| Note that su | $\mu[i+1 . . i+j+1], 0 \leq j \leq \min \{i, n-i\}\}$. |
| 1$]$ is an Abelian square centered at position $i$ iff |  | $\ell \in L_{i}$. After computing all $L_{i}$ 's, we store them in a two dimensional array $L$ of size $\left\lfloor\frac{n}{2}\right\rfloor \times n-1$ such that $L[\ell, i]=1$ if $\ell \in L_{i}$ and $L[\ell, i]=0$ otherwise. All entries of $L$ are initialized unmarked. Then, for each $1 \leq \ell \leq n-1$, all maximal Abelian repetitions of period $\ell$ can be computed by in $O(n)$ time, as follows. We scan the $\ell$ th row of $L$ from left to right for increasing $i=1, \ldots, n-1$, and if we encounter an unmarked entry $(\ell, i)$ such that $L[\ell, i]=1$, then we compute the largest non-negative integer $k$ such that $L[\ell, i+p \ell+1]=1$ for all $1 \leq p \leq k$ in $O(k)$ time, by skipping every $\ell-1$ entries in between. This gives us a maximal Abelian repetitions with period $\ell$


$L$| $\boldsymbol{i}$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 |
| 3 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 4 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 5 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Figure 4. The two dimensional array $L$ for string caaabababac. The maximal Abelian repetitions aaa of period 1 starting at position 3 is found by concatenating two Abelian squares represented by $L[1,2]$ and $L[1,3]$. The maximal Abelian repetition ababab of period 2 starting at position 4 is found by concatenating two Abelian squares represented by $L[2,5]$ and $L[2,7]$. The maximal Abelian repetition bababa of period 2 starting at position 5 is found by concatenating two Abelian squares represented by $L[2,6]$ and $L[2,8]$. Finally, the maximal Abelian repetition aababa of period 3 starting at position 3 is found from $L[3,5]$ (this is not extensible to the right). Every concatenation procedure (represented by an arrow) starts from an unmarked entry, and once an entry is involved in computation of a maximal Abelian repetition, it gets marked. This way the algorithm runs in time linear in the size of $L$, which is $O\left(n^{2}\right)$.
starting at position $i-\ell+1$ and ending at position $i+(k+1) \ell$. After computing the largest integer $k$, we mark the entries $L[\ell, \quad \leq p \leq k$ in $O(k)$ time. Since each unmarked entry of the $\ell t l$ accessed by a constant number of times, and sir from unmarked entries, it takes a total of $O(n)$ a total of $O\left(n^{2}\right)$ time for all $1 \leq \ell \leq\left\lfloor\frac{n}{2}\right\rfloor$. (See of the two dimensional array $L$ and how to comp from $L$ ).

What remains is how to compute the left and runs. If we compute the left and right hands $n$ repetitions, then it takes a total of $O\left(n^{3}\right)$ time due to left and right hands in a total of $O\left(n^{2}\right)$ time, we use the repetitions: For each $1 \leq i \leq n$, let $P_{i}$ be the set of each $\ell \in P_{i}$ there exists a maximal Abelian repetition w once and is
e starts only
ce, this takes
rete example
titions
belian belian te the belian nat for inning position is $i-\ell+1$. For any $1 \leq j \leq\left|P_{i}\right|$, let $\ell_{j}$ deno $P_{i}$. We process $\ell_{j}$ in increasing order of $j=1, \ldots,\left|P_{i}\right|$. Let $h_{j}$ denote the left hand of the Abelian run that is computed from the maximal Abelian repetition whose period is $\ell_{j}$ and beginning position is $i-\ell_{j}+1$. For any $1 \leq j<\left|P_{i}\right|$, assume that we have computed the length of the left hand $h_{j-1}$ of the maximal Abelian repetition beginning at position $i-\ell_{j-1}+1$. We are now computing the left hand $h_{j}$ of the next Abelian run. There are two cases to consider-

1. If $j=1$ or $\ell_{j-1}+h_{j-1} \leq \ell_{j}$, then we co Abelian repetition beginning at positior vector $P_{w\left[i-\ell_{j}-k . i-\ell_{j}\right]}$ for increasing $k$ fron $P_{w\left[i-\ell_{j}+1 . . i\right]}$. This takes $O\left(h_{j}\right)$ time. (See
2. If $\ell_{j-1}+h_{j-1}>\ell_{j}$, then

$$
P_{w\left[i-\ell_{j-1}-h_{j-1}+1 . . i-\ell_{j}\right]} \prec P_{w\left[i-\ell_{j-1}-h_{j-1}+1\right.}
$$

This implies that $h_{j} \geq \ell_{j-1}+h_{j-1}-\ell_{j}$.
 $P_{w\left[i-\ell_{j-1}-h_{j-1}+1 . . i-\ell_{j-1}\right]}$ in $O\left(\ell_{j}-\ell_{j-1}\right)$ time. Then, we compute the left hand $h_{j}$


## 4 Conclusions and future work

Abelian regularities on strings were initiated ly 60 's, and since then they have been extensively studied in $s$ new regularities on strings with respect to $A$ call Abelian covers and Abelian runs. Firstly, space algorithm to determine whether or n , strings, which we $O(n)$-time $O(\sigma)$ length $n$ over an alphabet of size $\sigma$ has an Abelian cover. As a consequence of this, we can compute the longest Abelian cover of $w$ in $O(n)$-time. Secondly, we showed an $O\left(n^{2}\right)$-time algorithm to compute an $O\left(n^{2}\right)$-space representation of all (possibly exponentially many) Abelian covers of a string of length $n$. Thirdly, we presented an $O\left(n^{2}\right)$-time
algorithm to compu
the maximum numb Our future work

- The algorithm given string of 1 in $o\left(n^{2}\right)$ time?
- The algorithm o a given string $w$ space. Can we c is the number of

string of length $n$. We also remarked that string of length $n$ is $\Omega\left(n^{2}\right)$.
o compute a shortest Abelian cover of a an we compute a shortest Abelian cover
$\left(n^{2}\right)$ time to compute all Abelian runs of to the two dimensional array $L$ of $\Theta\left(n^{2}\right)$ in $w$ in optimal $O(n+r)$ time, where $r$


4. L. J. Cummings and W. F. Smyth: Weak repetitions in strings. J. Combinatorial Mathematics and Combinatorial Computing, 24 1997, pp. 33-48.
5. R. C. Entringer and D. E. Jackson: On nonrepetitive sequences. J. Comb. Theory, Ser. A, 16(2) 1974, pp. 159-164.
6. P. Erdös: Some unsolved problems. Hungarian Academy of Sciences Mat. Kutató Intézet Közl, 6 1961, pp. 221-254.
7. G. Fici, T. Lecroq, A. Lefebvre, Élise Prieur-Gaston, and W. F. Smyth: Quasi-linear time computation of the Abelian periods of a word, in Proc. PSC 2012, 2012, pp. 103-110.
8. G. Fici, T. Lecroq, A. Lefebvre, and E. Prieur-Gaston: Computing Abelian periods in words, in Proc. PSC 2011, 2011, pp. 184-196.
9. T. Gagie, D. Hermelin, G. M. Landau, and O. Weimann: Binary jumbled pattern matching on trees and tree-like structures, in Proc. ESA 2013, 2013, pp. 517-528.
10. D. Hermelin, G. M. Landau, Y. Rabinovich, and O. Weimann: Binary jumbled pattern matching via all-pairs shortest paths. CoRR, abs/1401.2065 2014.
11. V. Keränen: Abelian squares are avoidable on 4 letters, in Proc. ICALP 1992, 1992, pp. 41-52.
12. T. Kociumaka, J. Radoszewski, and W. Rytter: Efficient indexes for jumbled pattern matching with constant-sized alphabet, in Proc. ESA 2013, 2013, pp. 625-636.
13. T. Kociumaka, J. Radoszewski, and W. Rytter: Fast algorithms for Abelian periods in words and greatest common divisor queries, in Proc. STACS 2013, 2013, pp. 245-256.
14. R. Kolpakov and G. Kucherov: Finding maximal repetitions in a word in linear time, in Proc. FOCS 1999, 1999, pp. 596-604.
15. Y. Li and W. F. Smyth: Computing the cover array in linear time. Algorithmica, 32(1) 2002, pp. 95-106.
16. T. M. Moosa and M. S. Rahman: Indexing permutations for binary strings. Inf. Process. Lett., 110(18-19) 2010, pp. 795-798.
17. R. Parikh: On context-free languages. J. ACM, 13(4) 1966, pp. 570-581.
18. P. A. B. Pleasants: Non-repetitive sequences. Mathematical Proceedings of the Cambridge Philosophical Society, 689 1970, pp. 267-274.
