k-Abelian Pattern Matching

Thorsten Ehlers Florin Manea Robert Mercaş Dirk Nowotka

Dependable Systems Group • **Department of Computer Science** • **KIEL University**

k-Abelian Equivalence

Basic notation

- $\Sigma = \{1, 2, \ldots, \sigma\}$: (finite) alphabet
- Σ^*, $\Sigma^{\leq k}$, $\Sigma^{=k}$: all finite words, of length $\leq k$, $= k$, resp., over Σ
- $|w|_t$: number of occurrences of t in w

k-Abelian Equivalence

Basic notation

- $\Sigma = \{1, 2, \ldots, \sigma\}$: (finite) alphabet
- Σ^*, $\Sigma^{\leq k}$, $\Sigma^{=k}$: all finite words, of length $\leq k$, $= k$, resp., over Σ
- $|w|_t$: number of occurrences of t in w

Words u and v, with $|u| = |v| = n$ are

- k-abelian equivalent, $u \equiv_k v$, if $|u|_t = |v|_t$ for all $t \in \Sigma^{\leq k}$,

k-Abelian Equivalence

Basic notation

- $\Sigma = \{1, 2, \ldots, \sigma\}$: (finite) alphabet
- Σ^*, $\Sigma^{\leq k}$, $\Sigma^{=k}$: all finite words, of length $\leq k$, $= k$, resp., over Σ
- $|w|_t$: number of occurrences of t in w

Words u and v, with $|u| = |v| = n$ are

- k-abelian equivalent, $u \equiv_k v$, if $|u|_t = |v|_t$ for all $t \in \Sigma^{\leq k}$,
- abelian equivalent, if $u \equiv_1 v$,
- equal, if $u \equiv_n v$.

k-Abelian Equivalence

Basic notation

- $\Sigma = \{1, 2, \ldots, \sigma\}$: (finite) alphabet
- Σ^*, $\Sigma^{\leq k}$, $\Sigma^{=k}$: all finite words, of length $\leq k$, $= k$, resp., over Σ
- $|w|_t$: number of occurrences of t in w

Words u and v, with $|u| = |v| = n$ are

- k-abelian equivalent, $u \equiv_k v$, if $|u|_t = |v|_t$ for all $t \in \Sigma^{\leq k}$,
- abelian equivalent, if $u \equiv_1 v$,
- equal, if $u \equiv_n v$.

Alternatively, u and v are k-abelian equivalent, if $u = v$ or $k < n$ and

- $|u|_t = |v|_t$ for all $t \in \Sigma^{=k}$ and
- $u[1 \ldots k-1] = v[1 \ldots k-1]$.

k-Abelian Pattern Matching

Basic notation

- Text: (long) word T of length n
- Pattern: (short) word P of length m

k-Abelian Pattern Matching

Basic notation

- Text: (long) word T of length n
- Pattern: (short) word P of length m

Find all occurrences of factors P' of length m in T such that

- $P' = P$ (pattern matching).
- $P' \equiv_k P$ (k-abelian pattern matching).

k-Abelian Pattern Matching

Basic notation

- Text: (long) word T of length n
- Pattern: (short) word P of length m

Find all occurrences of factors P' of length m in T such that

- $P' = P$ (pattern matching).
- $P' \equiv_k P$ (k-abelian pattern matching).

Assumptions

- computational model: RAM with logarithmic word size
- for every input word w of length ℓ: letters of w occur in $\{1, \ldots \ell\}$

Abelian Pattern Matching

Fact

Given P and T.

All occurrences of factors P' in T such that $P' \equiv_1 P$ can be found in $\mathcal{O}(n+m)$ time.

Abelian Pattern Matching

Fact

Given P and T.
All occurrences of factors P' in T such that $P' \equiv_1 P$ can be found in $\mathcal{O}(n+m)$ time.

Indeed,

- use sliding window of length m and update corresponding Parikh-vector in $\mathcal{O}(1)$ time, and
- monitor difference with Parikh-vector of P.

Some Data Structures

Consider w of length n over $\Sigma \subseteq \{1, \ldots, n\}$.

Some Data Structures

Consider w of length n over $\Sigma \subseteq \{1, \ldots, n\}$.

Suffix array Suf_w

- permutation of $\{1, \ldots, n\}$ with
- $Suf_w[i] = j$ if $w[j \ldots n]$ is i-th suffix in lexicographic order.

Some Data Structures

Consider w of length n over $\Sigma \subseteq \{1, \ldots, n\}$.

Suffix array Suf_w

- permutation of $\{1, \ldots, n\}$ with
- $Suf_w[i] = j$ if $w[j \ldots n]$ is i-th suffix in lexicographic order.

Longest Common Prefix array lcp_w

- array of length n with elements in $\{0, \ldots, n-1\}$ with
- $lcp_w[1] = 1$ and
- $lcp_w[r]$ length of longest common prefix
  of $w[Suf_w[r-1] \ldots n]$ and $w[Suf_w[r] \ldots n]$, for $r > 1$.

More on Data Structures

Lemma

Suf_w and lcp_w can be computed in $\mathcal{O}(n)$ time.

More on Data Structures

Lemma

Suf_w and lcp_w can be computed in $\mathcal{O}(n)$ time.

$LCP(i, j)$ query

What is the length of the longest common prefix of $w[i \ldots n]$ and $w[j \ldots n]$?

More on Data Structures

Lemma

Suf_w and lcp_w can be computed in $\mathcal{O}(n)$ time.

$LCP(i, j)$ query

What is the length of the longest common prefix of $w[i \ldots n]$ and $w[j \ldots n]$?

Lemma

After $\mathcal{O}(n)$ time preprocessing of lcp_w the $LCP(i, j)$ query can be answered in $\mathcal{O}(1)$ time.

k-Encoding

k-Encoding $\sharp(w, k)$

- let $S = \{w[i+1 \ldots i+k] \mid 0 \leq i \leq n-k\}$ set of length k factors,
- sort S lexicographically, and
- let $rank(i) = j$, if $w[i+1 \ldots i+k]$ is j-th element in S,

Note: Range of $\{rank(i)\}$ is ordered alphabet over encoding of length k factors.

k-Encoding

k-Encoding $\sharp(w,k)$

- let $S = \{w[i+1 \ldots i+k] \mid 0 \leq i \leq n-k\}$ set of length k factors,
- sort S lexicographically, and
- let $rank(i) = j$, if $w[i+1 \ldots i+k]$ is j-th element in S,

Note: Range of $\{rank(i)\}$ is ordered alphabet over encoding of length k factors.

- $\sharp(w,k)$ word of length $n-k+1$ over $\{1,\ldots,n-k+1\}$ with
- $\sharp(w,k)[i] = rank(i)$

k-Encoding

k-Encoding $\sharp(w,k)$

- let $S = \{w[i+1 \ldots i+k] \mid 0 \leq i \leq n-k\}$ set of length k factors,
- sort S lexicographically, and
- let $rank(i) = j$, if $w[i+1 \ldots i+k]$ is j-th element in S,

Note: Range of $\{rank(i)\}$ is ordered alphabet over encoding of length k factors.

- $\sharp(w,k)$ word of length $n-k+1$ over $\{1,\ldots,n-k+1\}$ with
- $\sharp(w,k)[i] = rank(i)$

Lemma

$\sharp(w,k)$ can be computed in $\mathcal{O}(n)$ time.

k-Encoding

k-Encoding $\sharp(w,k)$

- let $S = \{w[i+1 \ldots i+k] \mid 0 \leq i \leq n-k\}$ set of length k factors,
- sort S lexicographically, and
- let $rank(i) = j$, if $w[i+1 \ldots i+k]$ is j-th element in S,

Note: Range of $\{rank(i)\}$ is ordered alphabet over encoding of length k factors.

- $\sharp(w,k)$ word of length $n-k+1$ over $\{1,\ldots,n-k+1\}$ with
- $\sharp(w,k)[i] = rank(i)$

Lemma

$\sharp(w,k)$ can be computed in $\mathcal{O}(n)$ time.

Proof sketch.

- Identify in Suf_w contiguous groups of suffixes with common prefix of length k.
- Assign to these groups consecutive numbers from left to right.
- Ignore suffixes of length $< k$.

Offline k-Abelian Pattern Matching

Theorem

Given k, P and T.

All occurrences of factors P' in T, such that $P' \equiv_k P$, can be found in $\mathcal{O}(n+m)$ time.

Offline k-Abelian Pattern Matching

Theorem

Given k, P and T.
All occurrences of factors P' in T, such that $P' \equiv_k P$, can be found in $\mathcal{O}(n+m)$ time.

Proof sketch.

- Suppose $k < m < n$.
- Consider $w = T0P$ and construct $w' = \sharp(w, k)$ in $\mathcal{O}(n+m)$ time.

Offline k-Abelian Pattern Matching

Theorem

Given k, P and T.
All occurrences of factors P' in T, such that $P' \equiv_k P$, can be found in $\mathcal{O}(n+m)$ time.

Proof sketch.

- Suppose $k < m < n$.
- Consider $w = T0P$ and construct $w' = \sharp(w, k)$ in $\mathcal{O}(n+m)$ time.
- Construct data structure for LCP queries on w in $\mathcal{O}(n+m)$ time.
- $T' = w'[1 \ldots n-k+1]$ and $P' = w'[n+2 \ldots n+m-k+2]$.

Offline k-Abelian Pattern Matching

Theorem

Given k, P and T.
All occurrences of factors P' in T, such that $P' \equiv_k P$, can be found in $\mathcal{O}(n+m)$ time.

Proof sketch.

- Suppose $k < m < n$.
- Consider $w = T0P$ and construct $w' = \sharp(w, k)$ in $\mathcal{O}(n+m)$ time.
- Construct data structure for LCP queries on w in $\mathcal{O}(n+m)$ time.
- $T' = w'[1 \dots n-k+1]$ and $P' = w'[n+2 \dots n+m-k+2]$.
- Now, $T[i \dots i+m-1] \equiv_k P$ iff
  1. $T'[i \dots i+m-k-1] \equiv_1 P'$, and
  2. $T[i \dots i+k-2] = P[1 \dots k-1]$.

Offline k-Abelian Pattern Matching

Theorem

Given k, P and T.
All occurrences of factors P' in T, such that $P' \equiv_k P$, can be found in $\mathcal{O}(n+m)$ time.

Proof sketch.

- Suppose $k < m < n$.
- Consider $w = T0P$ and construct $w' = \sharp(w, k)$ in $\mathcal{O}(n+m)$ time.
- Construct data structure for LCP queries on w in $\mathcal{O}(n+m)$ time.
- $T' = w'[1 \ldots n-k+1]$ and $P' = w'[n+2 \ldots n+m-k+2]$.
- Now, $T[i \ldots i+m-1] \equiv_k P$ iff
  1. $T'[i \ldots i+m-k-1] \equiv_1 P'$, and
  2. $T[i \ldots i+k-2] = P[1 \ldots k-1]$.
- Algorithm: abelian search in T' for (1) in linear time;
  test for (2) in constant time using the LCP data structure.

Offline k-Abelian Pattern Matching (2)

Corollary

Given k and a word u of length n.
All factors of w that are k-abelian powers, can be identified in $\Theta((n-k)^2)$ time.

Idea: Reduction to finding abelian powers in $\sharp(w, k)$.

Offline k-Abelian Pattern Matching (2)

Corollary

Given k and a word u of length n.
All factors of w that are k-abelian powers, can be identified in $\Theta((n-k)^2)$ time.

Idea: Reduction to finding abelian powers in $\sharp(w, k)$.

Theorem

Given words u and v of length n.
The maximum k such that $u \equiv_k v$ can be found in $\mathcal{O}(n)$ time.

Note, that naïvely searching through all $1 \leq k \leq n$ takes $\mathcal{O}(n \log n)$ time.

Idea: Again, consider $w = u0v$ and its Suf_w and LCP data structures.

Online k-Abelian Pattern Matching

Preprocessing time

Given P. Time to build a data structure knowing P.

Query time

Given T letter by letter. Time to detect suffix P' with $P' \equiv_k P$ of prefix of T read so far.

Real-time k-abelian pattern matching, if query time is $\mathcal{O}(1)$.

Online k-Abelian Pattern Matching (2)

Theorem

The online k-abelian pattern matching problem can be solved in:

Preprocessing		*Query*
Time	*Space*	*Time*
$\mathcal{O}(m\|\Sigma\|)$	$\mathcal{O}(m\|\Sigma\|)$	$\mathcal{O}(1)$
$\mathcal{O}(m \log\log m)$	$\mathcal{O}(m)$	$\mathcal{O}(1)$
$\mathcal{O}(m)$ *expected*	$\mathcal{O}(m)$	$\mathcal{O}(1)$
$\mathcal{O}(m)$	$\mathcal{O}(m)$	$\mathcal{O}(\log\log\|\Sigma\|)$

Online extended-k-Abelian Pattern Matching

Words u and v, with $k < |u| = |v|$, are

- extended-k-abelian equivalent, if $|u|_t = |v|_t$ for all $t \in \Sigma^{=k}$.

Online extended-k-Abelian Pattern Matching

Words u and v, with $k < |u| = |v|$, are

- extended-k-abelian equivalent, if $|u|_t = |v|_t$ for all $t \in \Sigma^{=k}$.

Theorem

Given P and k for preprocessing.

The real-time extended-k-abelian pattern matching problem can be solved with $\mathcal{O}(m \log k)$ time and $\mathcal{O}(m)$ space for preprocessing.

— End of Talk —

Thank you for your attention!