k-Abelian Pattern Matching

Robert Mercaș

University of Kiel

Combinatorics and Algorithmics of Strings 2014

A joint work with Thorsten Ehlers, Florin Manea, and Dirk Nowotka

Definitions

Words u and v over Σ are *abelian equivalent* if $|u|_a = |v|_a$ for every $a \in \Sigma$.

Definition

Two words u and v are *k-abelian equivalent*:

- if $|u|_t = |v|_t$ for every word t of length at most k.
- if $|u|_t = |v|_t$ for every word t of length k, $\mathrm{pref}_{k-1}(u) = \mathrm{pref}_{k-1}(v)$ and $\mathrm{suff}_{k-1}(u) = \mathrm{suff}_{k-1}(v)$.

We denote this by $u \equiv_k v$.

A *k-abelian nth power* is a word $u_1 u_2 \ldots u_n$, where $u_1, u_2 \ldots, u_n$ are k-abelian equivalent.

Example k-abelian equivalence

$$u \equiv_3 v$$

$u = abbbaaaba$, $v = abaaabbba$, and $k = 3$

$$(u,3) = \{(aaa,1),(aab,1),(aba,1),(abb,1),(bba,1),(bbb,1)\} = (w,3)$$
$$(u,2) = \{(aa,2),(ab,2),(ba,2),(bb,2)\} = (w,2)$$
$$(u,1) = \{(a,5),(b,4)\} = (w,1)$$

$$x \not\equiv_2 v$$

$x = aba$, $y = bab$, and $k = 2$

$$(x,2) = \{(ab,1),(ba,1)\} = (y,2)$$
$$(x,1) = \{(a,2),(b,1)\} \neq \{(a,1),(b,2)\} = (y,2)$$

why?

[Halava et al.: Local Squares, Periodicity and Finite Automata, 2011]

- 2-abelian cubes avoidable on binary alphabet, 3-abelian squares avoidable on ternary alphabet
- complexity function is somewhere between that of equality and classical abelian operation
- variations of Morse-Hedlund Theorem for k-abelian equivalence

Classical Results

Theorem

For a pattern $P \in \Sigma^*$ and a text $T \in \Sigma^*$, we can identify all factors of T that are abelian equivalent to P in $\mathcal{O}(|T| + |P|)$ time.

Theorem

For a word $w \in \Sigma^*$, we can identify all factors of w that are abelian repetitions in $\mathcal{O}(|w|^2)$ time.

Tool: the $\#(w,k)$ encoding

$$w = aabbccaabbcc \text{ and } k = 2$$

7	1	8	2	9	3	10	4	12	6	11	5
a	a	a	a	b	b	b	b	c	c	c	c
a	a	b	b	b	b	c	c		a	c	c
b	b	b	b	c	c	c	c		a		a
b	b	c	c	c	c		a		b		a
c	c	c	c		a		a		b		b
c	c		a		a		b		c		b
	a		a		b		b		c		c
	a		b		b		c				c
	b		b		c		c				
	b		c		c						
	c		c								
	c										

7	1	8	2	9	3	10	4	12	6	11	5
1	1	2	2	3	3	4	4	-	5	6	6

$$\#(w,2) = 23456512346$$

Properties of encodings

Lemma

Let $w \in \Sigma^*$ be a word of length n. We can compute $\#(w, k)$ in $\mathcal{O}(n)$ time.

Lemma

Let $u, v \in \Sigma^*$ be two words of length n. If $u \equiv_k v$ for some k, then

$$\begin{aligned} u[1..k-1] &= v[1..k-1], \text{ and} \\ \#(u,k) &\equiv_1 \#(v,k). \end{aligned}$$

Testing k-equivalence

Lemma

Let $u, v \in \Sigma^*$ be two words of length n and k be a positive integer with $1 \leq k \leq n$. We can decide whether $u \equiv_k v$ in $\mathcal{O}(n)$ time.

Proof.

Construct $w = u0v$, and compute Suf_w and $\#(w, k)$.
Set $u' = \#(w, k)[1..n-k+1]$ and $v' = \#(w, k)[n+2..2n-k+2]$.
The fact that u' and v' contain exactly the same letters is equivalent to them having exactly the same multi-set of factors of length k.

Then $u \equiv_k u$ if and only if

$$\begin{aligned} u[1..k-1] &= v[1..k-1], \text{ and} \\ u' &\equiv_1 v'. \end{aligned}$$

□

Pattern Matching Results

Remark

The k-abelian pattern matching problem can be reduced to the classical abelian pattern matching problem.

Theorem

For a pattern $P \in \Sigma^*$, a text $T \in \Sigma^*$, and an integer k, we can identify all factors of T that are k-abelian equivalent to P in $\mathcal{O}(|T| + |P|)$ time.

Theorem

For a word $w \in \Sigma^*$ and an integer k, we can identify all k-abelian repetitions in w in $\mathcal{O}(|w|^2)$ time.

Another problem

Theorem

Given two words of same length n, we can find the largest positive integer k such that the words are k-abelian equivalent linear time $\mathcal{O}(n)$.

Construct the word $w = u0v$, its Suf_w, and $LCPref_w$ data structures. Set $u' = \#(w,k)[1..n-k+1]$ and $v' = \#(w,k)[n+2..2n-k+2]$.

Set $\ell = \min(\max_{pref}(u',v'), \max_{suff}(u',v')) + 1$. Then $k \leq \ell$.

Going through Suf_w we do $LCPref_w$ queries for the i^{th} suffix of u (in lexicographic order) and the i^{th} suffix of v. Let ℓ' be the minimum value such that two such suffixes share a common prefix of length exactly ℓ', while both have length at least $\ell' + 1$.

Then $k = \min\{\ell', \ell\}$.

Online problem

Problem

Preprocess a word $P \in \Sigma^*$ and a positive integer k such that when given a text $T \in \Sigma^*$, in letter by letter manner, to be able to answer, at each moment, queries asking whether the part of T read so far ends with a factor which is k-abelian equivalent to P or not.

Construct the $\#(P, k-1)$ and $\#(P, k)$ encodings over $\{1, \ldots, \ell_2, \ldots, \ell_1\}$.

Compute $L = \{(i, a, j) \mid 1 \leq i \leq \ell_1, 1 \leq j \leq \ell_2, a \in \Sigma, \text{ and } f_1[i]a = f_2[j]\}$, where $f_1[i]$ ($f_2[i]$) is the factor of length $k-1$ (k) of P, whose position in the lexicographically ordered set of all factors of length $k-1$ (k) of P is i.

Online Solution

Different implementations for L

- An $m \times \sigma$ table M, with $M[i][a] = j$ if and only if $(i, a, j) \in L$.
- A hash table, using perfect hashing.
- A van Emde Boas tree associating to each i a pair $(a, \cdot)$, if this exists.

Compute for each $1 \leq j \leq \ell_2$ the values $suf[j], pref[j] \leq \ell_1$, such that $suf[j] = i$ if $f_2[j] = af_1[i]$ for $a \in \Sigma$, while $pref[j] = i$ if $f_2[j] = f_1[i]a$.

If last read part of T is $\#(P', k) = j_1 \ldots j_{m-k} j_{m-k+1}$ and we read letter a, then the new last part is $\#(P'', k) = j_2 \ldots j_{m-k} j_{m-k+1} suf[j_{m-k+1}]a$.

- If $(suf[j_{m-k+1}], a, \cdot) \in L$, then use real-time abelian pattern matching between $\#(P'', k)$ and $\#(P, k)$.
- If $(suf[j_{m-k+1}], a, \cdot) \notin L$, then use real-time pattern matching between the suffix of length $k-1$ of the read text and $P[1..k-1]$.

thus...

Theorem

Given a pattern $P \in \Sigma^m$ for $|\Sigma| = \sigma$, and a positive integer k, the online k-abelian pattern matching problem can be solved in:

- $\mathcal{O}(m\sigma)$ *preprocessing time,* $\mathcal{O}(m\sigma)$ *space, and* $\mathcal{O}(1)$ *query time.*
- $\mathcal{O}(m \log\log m)$ *preprocessing time,* $\mathcal{O}(m)$ *space, and* $\mathcal{O}(1)$ *query time.*
- $\mathcal{O}(m)$ *expect. preprocessing time,* $\mathcal{O}(m)$ *space, and* $\mathcal{O}(1)$ *query time.*
- $\mathcal{O}(m)$ *preprocessing time,* $\mathcal{O}(m)$ *space, and* $\mathcal{O}(\log\log\sigma)$ *query time.*

Extended-k-abelian equivalence

Problem

Preprocess a word $P \in \Sigma^*$ and a positive integer k such that when given a text $T \in \Sigma^*$, in letter by letter manner, to be able to answer, at each moment, queries asking whether the part of T read so far ends with a factor which is extended-k-abelian equivalent to P or not.

$$x \not\equiv_2 v$$

$$x = aba,\ y = bab, \text{ and } k = 2$$

$$(x, 2) = \{(ab, 1), (ba, 1)\} = (y, 2)$$
$$(x, 1) = \{(a, 2), (b, 1)\} \neq \{(a, 1), (b, 2)\} = (y, 2)$$

$$x \equiv_2^e v$$

Online Solution for extended case

Extra tools:

- Suffix tree for P
- Associate to a node of the suffix tree of P the factor $P[i..j]$ iff the path from the root to that node is labelled with $P[i..j]$.
- Suffix links (from the node with label aX go to the node with label X)

1. Let N_1 be the lowest explicit ancestor of N the node for $P[i..i+\ell-1]$.
2. If exists and edge labeled $a = T[j+1]$ from N to M, then $T[1..j+1]$ has the suffix $P[i..i+\ell-1]a$ and we update the current node to M.
3. Otherwise, let N_2 be the target-node of the suffix link of N_1.Advance along the edge leaving N_2, and update each node we find to be the current N_2 until no longer possible. The current N_2 is the lowest explicit ancestor of the node corresponding to $P[i+1..i+\ell-1]$.
4. Update $i = i+1$ and $N = N_2$, and restart from step 2.

thus...

Theorem

Given a pattern $P \in \Sigma^*$ of length n for $|\Sigma| = \sigma$, and a positive integer k, the online extended-k-abelian pattern matching problem can be solved in:

- $\mathcal{O}(m\sigma)$ *preprocessing time,* $\mathcal{O}(m\sigma)$ *space, and* $\mathcal{O}(n)$ *time.*
- $\mathcal{O}(m \log\log m)$ *preprocessing time,* $\mathcal{O}(m)$ *space, and* $\mathcal{O}(n)$ *time.*
- $\mathcal{O}(m)$ *expected preprocessing time,* $\mathcal{O}(m)$ *space, and* $\mathcal{O}(n)$ *time.*
- $\mathcal{O}(m)$ *preprocessing time,* $\mathcal{O}(m)$ *space, and* $\mathcal{O}(n \log\log \sigma)$ *time.*

Real-time solution for extended case

Idea: report only the factors of T that are extended-k-abelian equivalent to P (whenever a length k factor of P is found).

Lemma (GawNicLew14)

We can preprocess a word $P \in \Sigma^*$ in $\mathcal{O}(|P| \log k)$ time and linear space such that, for each i and j with $j - i \leq k$, we can return in $\mathcal{O}(1)$ time the (explicit or implicit) node of the suffix tree of P corresponding to $P[i..j]$.

Use a queue to add the current letter and 2 more operations:

1. dequeue and check whether the current factor of P be extended;
2. update the P-suffix and its representation accordingly, dequeue, and check whether the new current factor of P can be extended.

Enough, since not being able to extended for some length ℓ, the number of suffixes of factors of P we have to check until reaching again length ℓ equals the number of letters read between these two moments

thus...

Theorem

Given a pattern $P \in \Sigma^*$ of length n for $|\Sigma| = \sigma$, and an integer k, the real-time extended-k-abelian pattern matching problem can be solved in:

- $\mathcal{O}(m(\sigma + \log k))$ *preproc. time,* $\mathcal{O}(m\sigma)$ *space, and* $\mathcal{O}(1)$ *query time.*
- $\mathcal{O}(m(\log(k \log m))$ *preproc. time,* $\mathcal{O}(m)$ *space, and* $\mathcal{O}(1)$ *query time.*
- $\mathcal{O}(m \log k)$ *expec. preproc. time,* $\mathcal{O}(m)$ *space, and* $\mathcal{O}(1)$ *query time.*
- $\mathcal{O}(m \log k)$ *preproc. time,* $\mathcal{O}(m)$ *space, and* $\mathcal{O}(\log \log k)$ *query time.*

Downside: index structures for k-abelian pattern matching

Idea: construct $\#(T, k-1)$ and $\#(T, k)$ and $L = \{(i, a, j)\}$, as before.

Find a location of $P[1..k-1]$ in T in $\mathcal{O}(k)$ time.
Reading $P[k..m]$ letter by letter (checking against L) we produce $\#(P, k)$.
The problem is reduced to producing an index of $\#(T, k)$, useful to check efficiently whether a factor is abelian equivalent to $\#(P, k)$.

The preprocessing time for building an index for the k-abelian pattern matching problem is $\mathcal{O}(n)$ expected or $\mathcal{O}(n \log\log n)$ deterministic time.
The query time is $\mathcal{O}(n + m - k + 1)$.

Thank you!

Special thanks to
Prof. Dirk Nowotka!