

Jumbled String Matching: Motivations, Variants, Algorithms

Zsuzsanna Lipták

University of Verona (Italy)

Workshop

“Combinatorial structures for sequence analysis in bioinformatics”

Milano-Bicocca, 27 Nov 2013

Jumbled String Matching

Parikh vectors:

Given string t over constant-size ordered alphabet Σ , with $|\Sigma| = \sigma$.

The **Parikh vector** $p(t)$ counts the multiplicity of characters in t .

Ex.: $p(aabcac) = (3, 1, 2)$.

Jumbled String Matching

Parikh vectors:

Given string t over constant-size ordered alphabet Σ , with $|\Sigma| = \sigma$.

The **Parikh vector** $p(t)$ counts the multiplicity of characters in t .

Ex.: $p(aabcac) = (3, 1, 2)$.

(a.k.a. jumbled string = compomer = composition = ...)

Problem Statement

JUMBLED STRING MATCHING

Given string s of length n , and query Parikh vector $q \in \mathbb{N}^\sigma$.

Find all occurrences of substrings t of s s.t. $p(t) = q$.

Ex.: $\Sigma = \{a, b, c\}$, query $q = (3, 1, 2)$

b b a c a c c a b a b b a b c c a a a c

Problem Statement

JUMBLED STRING MATCHING

Given string s of length n , and query Parikh vector $q \in \mathbb{N}^\sigma$.

Find all occurrences of substrings t of s s.t. $p(t) = q$.

Ex.: $\Sigma = \{a, b, c\}$, query $q = (3, 1, 2)$

$b b a c$ $a c c a b a$ $b b$ $a b c c a a a$ c

= any permutation of $abcac$

Problem Statement

JUMBLED STRING MATCHING

Given string s of length n , and query Parikh vector $q \in \mathbb{N}^\sigma$.

Find all occurrences of substrings t of s s.t. $p(t) = q$.

Ex.: $\Sigma = \{a, b, c\}$, query $q = (3, 1, 2)$

$b b a c$ $a c c a b a$ $b b$ $a b c c a a a$ c

= any permutation of $abcac$ = any jumble of $abcac$

Problem Statement

JUMBLED STRING MATCHING

Given string s of length n , and query Parikh vector $q \in \mathbb{N}^\sigma$.

Find all occurrences of substrings t of s s.t. $p(t) = q$.

Ex.: $\Sigma = \{a, b, c\}$, query $q = (3, 1, 2)$

$b \ b \ a \ c \ a \ c \ c \ a \ b \ a \ b \ b \ a \ b \ c \ c \ a \ a \ a \ c$

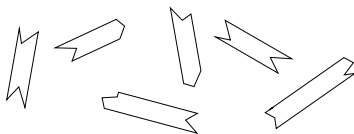
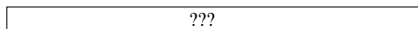
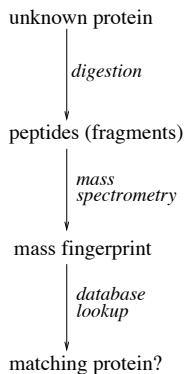
= any permutation of $abcac$ = any **jumble** of $abcac$

a.k.a. permutation matching, Parikh vector matching, abelian matching

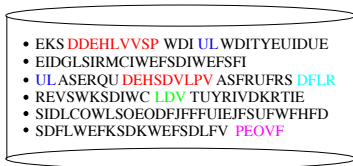
Motivations, Applications

- Mass spectrometry
- Gene clusters
- Motif search in graphs and trees
- Filter for exact pattern matching

Protein identification with mass spectrometry (here: PMF)



{154, 223, 317, 371, 748, 991}

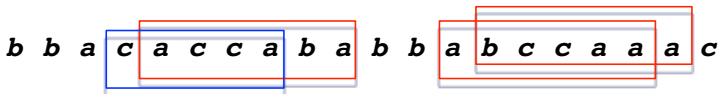


Takes advantage of different molecular masses of “characters” (AAs, nucleotides, ...)

Modelling sample identification with MS

Every character has a mass: $\mu : \Sigma \rightarrow \mathbb{R}^+$ mass function, $\mu(t) = \sum_i \mu(t_i)$.

Ex: $\Sigma = \{a, b, c\}$ with $\mu(a) = 2, \mu(b) = 3, \mu(c) = 5$. Query $M = 19$



Actually we can also look for all Parikh vectors with query mass $M!$
 \implies Jumbled String Matching!

Application 2: Gene clusters

Given: k genomes

Find: maximal blocks consisting of same genes

1 2 **3 4 3 5 1 5 4** 7 2 5

5 3 1 4 4 7 1 7 2 2 1 3

1 2 6 7 **5 1 3 5 4** 2 2 5

gene cluster: $\{1, 3, 4, 5\}$

Application 2: Gene clusters

Given: k genomes

Find: maximal blocks consisting of same genes

1 2 **3 4 3 5 1 5 4** 7 2 5

5 3 1 4 4 7 1 7 2 2 1 3

1 2 6 7 **5 1 3 5 4** 2 2 5

gene cluster: $\{1, 3, 4, 5\}$

Caveat: Problem slightly different (so far).

1. Simple solutions

Window algorithm

Jumbled string matching query $q = (3, 1, 2)$

b b a c a c c a b a b b a b c c a a a c

- sliding window: either fixed-size ($m = |q| = \sum_i q_i$), or variable-size

Window algorithm

Jumbled string matching query $q = (3, 1, 2)$

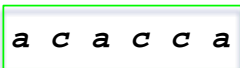
b **b a c a c c** **a b a b b a b c c a a a c**

- sliding window: either fixed-size ($m = |q| = \sum_i q_i$), or variable-size

Window algorithm

Jumbled string matching query $q = (3, 1, 2)$

b b a c a c c a b a b b a b c c a a a c




- sliding window: either fixed-size ($m = |q| = \sum_i q_i$), or variable-size

Window algorithm

Jumbled string matching query $q = (3, 1, 2)$

b b a c a c c a b a b b a b c c a a a c




- sliding window: either fixed-size ($m = |q| = \sum_i q_i$), or variable-size

Window algorithm

Jumbled string matching query $q = (3, 1, 2)$

b b a c a c c a b a b b a b c c a a a c

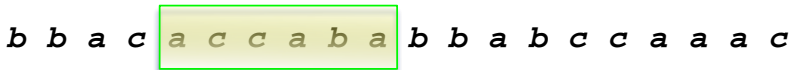


- sliding window: either fixed-size ($m = |q| = \sum_i q_i$), or variable-size

Window algorithm

Jumbled string matching query $q = (3, 1, 2)$

b b a c a c c a b a b b a b c c a a a c



- sliding window: either fixed-size ($m = |q| = \sum_i q_i$), or variable-size
- worst-case optimal for one query: $O(n)$ time, $O(\sigma)$ additional space.

Indexed jumbled string matching

What about many queries? \rightsquigarrow **indexed** version of problem

simple solutions (K queries):

1. no index: $O(Kn)$ query time
2. store all: $O(n^2)$ index size, $O(K \log n)$ query time.

Indexed jumbled string matching

What about many queries? \rightsquigarrow **indexed** version of problem

simple solutions (K queries):

1. no index: $O(Kn)$ query time
2. store all: $O(n^2)$ index size, $O(K \log n)$ query time.

For exact string matching, elaborate solutions exist:
suffix trees, suffix arrays, ...

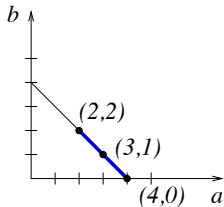
- index has $O(n)$ size (size of text)
- construction time and space $O(n)$
- query time $O(m)$ (size of query) – so $O(Km)$ for K queries

2. Binary alphabets

Binary alphabets: Interval property

Lemma

If $(x, m - x)$ and $(y, m - y)$ both occur in s , then so does $(z, m - z)$ for any $x \leq z \leq y$.



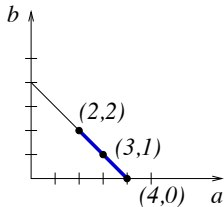
b a b a a b a a b b **a a a a** b a

$\text{amin}(4) = 2$ $\text{amax}(4) = 4$

Binary alphabets: Interval property

Lemma

If $(x, m - x)$ and $(y, m - y)$ both occur in s , then so does $(z, m - z)$ for any $x \leq z \leq y$.



b a b a

$$\mathit{amin}(4) = 2$$

a b a a b b

a a a a

$$\mathit{amax}(4) = 4$$

Corollary

All sub-Parikh vectors of s of length m build a set $\{(x, m - x) : \mathit{amin}(m) \leq x \leq \mathit{amax}(m)\}$.

Binary alphabets: Interval algorithm for decision queries

- **Index:** Table of $\text{amin}(m)$ and $\text{amax}(m)$, for $1 \leq m \leq n$
size $O(n)$
- **Query** (x, y) with occurs in s iff $\text{amin}(x + y) \leq x \leq \text{amax}(x + y)$.
- Query time $O(1)$.

m	amin	amax
...		
4	2	4
...		

query (3, 1) — YES

query (1, 3) — NO

Construction of index

Goal

Given a binary string of length n , find, for all $1 \leq m \leq n$, the minimum (maximum) number of a 's in a window of size m .

- $O(n^2)$ time—Cicalese, Fici, L. (PSC 2009)
- $O(n^2 / \log n)$ time
—Burcsi, Cicalese, Fici, L. (FUN 2010); Moosa, Rahman (IPL 2010)
- $O(n^2 / \log^2 n)$ time in word-RAM model
—Moosa, Rahman (JDA 2012)
- approximate index with one-sided error in $O(n^{1+\epsilon})$ time
—Cicalese, Laber, Weimann, Yuster (CPM 2012)
- Corner Index: construction time and index size depend on $r =$
runlength enc. of s
—Badkobeh, Fici, Kroon, L. (IPL 2013); Giaquinta, Grabowski (IPL 2013)

Index for binary JPM

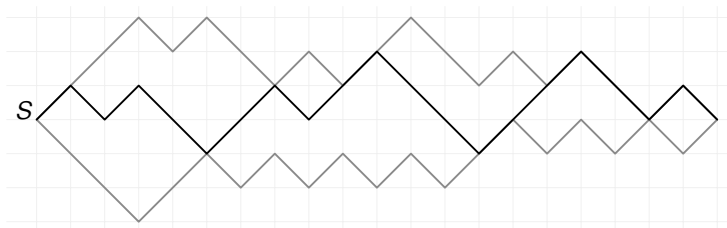


Figure : $/ = a$, $\backslash = b$, $s = ababbaabaabbbbaabbab$. Verticals are fixed length sub-Parikh vectors. The region is the **Parikh set** of w .

Construction of index

Goal

Given a binary string of length n , find, for all $1 \leq m \leq n$, the minimum (maximum) number of a 's in a window of size m .

- $O(n^2)$ time—Cicalese, Fici, L. (PSC 2009)
- $O(n^2 / \log n)$ time
—Burcsi, Cicalese, Fici, L. (FUN 2010); Moosa, Rahman (IPL 2010)
- $O(n^2 / \log^2 n)$ time in word-RAM model
—Moosa, Rahman (JDA 2012)
- approximate index with one-sided error in $O(n^{1+\epsilon})$ time
—Cicalese, Laber, Weimann, Yuster (CPM 2012)
- Corner Index: construction time and index size depend on $r =$
runlength enc. of s
—Badkobeh, Fici, Kroon, L. (IPL 2013); Giaquinta, Grabowski (IPL 2013)

Construction of index

Goal

Given a binary string of length n , find, for all $1 \leq m \leq n$, the minimum (maximum) number of a 's in a window of size m .

- $O(n^2)$ time—Cicalese, Fici, L. (PSC 2009)
- $O(n^2 / \log n)$ time
—Burcsi, Cicalese, Fici, L. (FUN 2010); Moosa, Rahman (IPL 2010)
- $O(n^2 / \log^2 n)$ time in word-RAM model
—Moosa, Rahman (JDA 2012)
- approximate index with one-sided error in $O(n^{1+\epsilon})$ time
—Cicalese, Laber, Weimann, Yuster (CPM 2012)
- Corner Index: construction time and index size depend on $r =$
runlength enc. of s
—Badkobeh, Fici, Kroon, L. (IPL 2013); Giaquinta, Grabowski (IPL 2013)

Open problem: Faster construction of index?

3. General alphabets: Jumping Algorithm

Jumping Algorithm for Jumbled String Matching

Cicalese, Fici, L. (PSC 2009, FUN 2010)

Recall the window algorithm.

fixed size window \rightarrow variable size window

$q = (312)$

b b a c a c c a b a b b a b c c a a a c

Jumping Algorithm for Jumbled String Matching

Cicalese, Fici, L. (PSC 2009, FUN 2010)

Recall the window algorithm.

fixed size window \rightarrow variable size window

$q = (312)$

b b a c a c c a b a b b a b c c a a a c

Jumping Algorithm for Jumbled String Matching

Cicalese, Fici, L. (PSC 2009, FUN 2010)

Recall the window algorithm.

fixed size window \rightarrow variable size window

$q = (312)$

b b a c a c c a b a b b a b c c a a a c

Jumping Algorithm for Jumbled String Matching

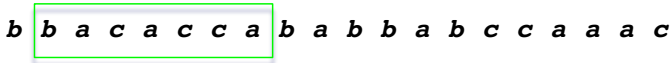
Cicalese, Fici, L. (PSC 2009, FUN 2010)

Recall the window algorithm.

fixed size window \rightarrow variable size window

$q = (312)$

b b a c a c c a b a b b a b c c a a a c



Jumping Algorithm for Jumbled String Matching

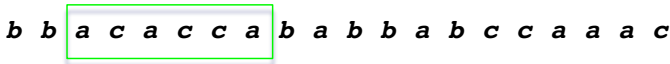
Cicalese, Fici, L. (PSC 2009, FUN 2010)

Recall the window algorithm.

fixed size window \rightarrow variable size window

$q = (312)$

b b a c a c c a b a b b a b c c a a a c



Jumping Algorithm for Jumbled String Matching

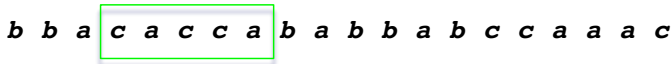
Cicalese, Fici, L. (PSC 2009, FUN 2010)

Recall the window algorithm.

fixed size window \rightarrow variable size window

$q = (312)$

b b a c a c c a b a b b a b c c a a a c



Jumping Algorithm for Jumbled String Matching

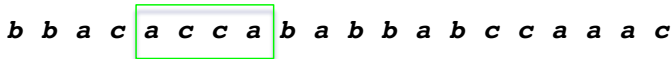
Cicalese, Fici, L. (PSC 2009, FUN 2010)

Recall the window algorithm.

fixed size window \rightarrow variable size window

$q = (312)$

b b a c a c c a b a b b a b c c a a a c



Jumping Algorithm for Jumbled String Matching

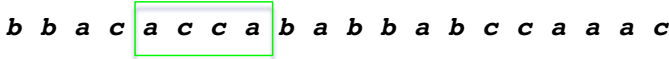
Cicalese, Fici, L. (PSC 2009, FUN 2010)

Recall the window algorithm.

fixed size window \rightarrow variable size window

$q = (312)$

b b a c a c c a b a b b a b c c a a a c



The Jumping Algorithm simulates these moves by jumps.

Jumping algorithm: update rules

$q = (312)$

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
L	↓																				
		b	b	a	c	a	c	c	a	b	a	b	b	a	b	c	c	a	a	a	c
a	0	0	1	1	2	2	2	3	3	4	4	4	5	5	5	5	6	7	8	8	
b	1	2	2	2	2	2	2	3	3	4	5	5	6	6	6	6	6	6	6	6	
c	0	0	0	1	1	2	3	3	3	3	3	3	3	3	4	5	5	5	5	6	

Jumping algorithm: update rules

$q = (312)$

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
L	↓								↓												
		b	b	a	c	a	c	c	a	b	a	b	b	a	b	c	c	a	a	a	c
a	0	0	1	1	2	2	2	3	3	4	4	4	5	5	5	5	6	7	8	8	
b	1	2	2	2	2	2	2	3	3	4	5	5	6	6	6	6	6	6	6	6	
c	0	0	0	1	1	2	3	3	3	3	3	3	3	3	4	5	5	5	5	6	

Jumping algorithm: update rules

$q = (312)$

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
L	↓								↓												
		b	b	a	c	a	c	c	a	b	a	b	b	a	b	c	c	a	a	a	c
a	0	0	1	1	2	2	2	3	3	4	4	4	5	5	5	5	6	7	8	8	
b	1	2	2	2	2	2	2	3	3	4	5	5	6	6	6	6	6	6	6	6	
c	0	0	0	1	1	2	3	3	3	3	3	3	3	3	4	5	5	5	5	6	

Jumping algorithm: update rules

$q = (312)$

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
$L \downarrow$									$R \downarrow$												
	b	b	a	c	a	c	c	a	b	a	b	b	a	b	c	c	a	a	a	c	
a	0	0	1	1	2	2	2	3	3	4	4	4	5	5	5	5	6	7	8	8	
b	1	2	2	2	2	2	2	3	3	4	5	5	6	6	6	6	6	6	6	6	
c	0	0	0	1	1	2	3	3	3	3	3	3	3	3	4	5	5	5	5	6	

update R : $R \leftarrow \max_{a \in \Sigma} \underbrace{\text{select}_a(\text{rank}_a(L) + q_a)}_{\text{necessary char's}}$

Jumping algorithm: update rules

$q = (312)$

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
				L				R												
	b	b	a	c	a	c	c	a	b	a	b	b	a	b	c	c	a	a	a	c
a	0	0	1	1	2	2	2	3	3	4	4	4	5	5	5	5	6	7	8	8
b	1	2	2	2	2	2	2	3	3	4	5	5	6	6	6	6	6	6	6	6
c	0	0	0	1	1	2	3	3	3	3	3	3	3	3	4	5	5	5	5	6

update R : $R \leftarrow \max_{a \in \Sigma} \underbrace{\text{select}_a(\text{rank}_a(L) + q_a)}_{\text{necessary char's}}$

Jumping algorithm: update rules

$q = (312)$

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
					$L \downarrow$				$R \downarrow$												
		b	b	a	c	a	c	c	a	b	a	b	b	a	b	c	c	a	a	a	c
a	0	0	1	1	2	2	2	3	3	4	4	4	5	5	5	5	6	7	8	8	
b	1	2	2	2	2	2	2	2	3	3	4	5	5	6	6	6	6	6	6	6	6
c	0	0	0	1	1	2	3	3	3	3	3	3	3	3	4	5	5	5	5	6	

update R : $R \leftarrow \max_{a \in \Sigma} \underbrace{\text{select}_a(\text{rank}_a(L) + q_a)}_{\text{necessary char's}}$

update L : $L \leftarrow \max_{a \in \Sigma} \underbrace{\text{select}_a(\text{rank}_a(R) - q_a)}_{\text{unnecessary char's}} \text{ (no match),}$

Jumping algorithm: update rules

$q = (312)$

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
					$L \downarrow$						$R \downarrow$										
		b	b	a	c	a	c	c	a	b	a	b	b	a	b	c	c	a	a	a	c
a	0	0	1	1	2	2	2	2	3	3	4	4	4	5	5	5	5	6	7	8	8
b	1	2	2	2	2	2	2	2	3	3	4	5	5	6	6	6	6	6	6	6	6
c	0	0	0	1	1	2	3	3	3	3	3	3	3	3	3	4	5	5	5	5	6

update R : $R \leftarrow \max_{a \in \Sigma} \underbrace{\text{select}_a(\text{rank}_a(L) + q_a)}_{\text{necessary char's}}$

update L : $L \leftarrow \max_{a \in \Sigma} \underbrace{\text{select}_a(\text{rank}_a(R) - q_a)}_{\text{unnecessary char's}} \text{ (no match),}$

Jumping algorithm: update rules

$q = (312)$

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
						$L \downarrow$					$R \downarrow$										
		b	b	a	c	a	c	c	a	b	a	b	b	a	b	c	c	a	a	a	c
a	0	0	1	1	2	2	2	2	3	3	4	4	4	5	5	5	5	6	7	8	8
b	1	2	2	2	2	2	2	2	3	3	4	5	5	6	6	6	6	6	6	6	6
c	0	0	0	1	1	2	3	3	3	3	3	3	3	3	4	5	5	5	5	6	6

update R : $R \leftarrow \max_{a \in \Sigma} \underbrace{\text{select}_a(\text{rank}_a(L) + q_a)}_{\text{necessary char's}}$

update L : $L \leftarrow \max_{a \in \Sigma} \underbrace{\text{select}_a(\text{rank}_a(R) - q_a)}_{\text{unnecessary char's}}$ (no match),

$L \leftarrow L + 1$ (match).

Jumping algorithm: update rules

$q = (312)$

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
						L ↓								R ↓							
		b	b	a	c	a	c	c	a	b	a	b	b	a	b	c	c	a	a	a	c
a	0	0	1	1	2	2	2	3	3	4	4	4	5	5	5	5	6	7	8	8	
b	1	2	2	2	2	2	2	3	3	4	5	5	6	6	6	6	6	6	6	6	
c	0	0	0	1	1	2	3	3	3	3	3	3	3	3	4	5	5	5	5	6	

update R : $R \leftarrow \max_{a \in \Sigma} \underbrace{\text{select}_a(\text{rank}_a(L) + q_a)}_{\text{necessary char's}}$

update L : $L \leftarrow \max_{a \in \Sigma} \underbrace{\text{select}_a(\text{rank}_a(R) - q_a)}_{\text{unnecessary char's}}$ (no match),

$L \leftarrow L + 1$ (match).

Jumping algorithm: update rules

$q = (312)$

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
												$L \downarrow$	$R \downarrow$								
		b	b	a	c	a	c	c	a	b	a	b	b	a	b	c	c	a	a	a	c
a	0	0	1	1	2	2	2	2	3	3	4	4	4	5	5	5	5	6	7	8	8
b	1	2	2	2	2	2	2	2	3	3	4	5	5	6	6	6	6	6	6	6	6
c	0	0	0	1	1	2	3	3	3	3	3	3	3	3	4	5	5	5	5	5	6

update R : $R \leftarrow \max_{a \in \Sigma} \underbrace{\text{select}_a(\text{rank}_a(L) + q_a)}_{\text{necessary char's}}$

update L : $L \leftarrow \max_{a \in \Sigma} \underbrace{\text{select}_a(\text{rank}_a(R) - q_a)}_{\text{unnecessary char's}}$ (no match),

$L \leftarrow L + 1$ (match).

Jumping algorithm: update rules

$q = (312)$

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
												$L \downarrow$							$R \downarrow$		
		b	b	a	c	a	c	c	a	b	a	b	b	a	b	c	c	a	a	a	c
a	0	0	1	1	2	2	2	2	3	3	4	4	4	5	5	5	5	6	7	8	8
b	1	2	2	2	2	2	2	2	3	3	4	5	5	6	6	6	6	6	6	6	6
c	0	0	0	1	1	2	3	3	3	3	3	3	3	3	4	5	5	5	5	5	6

update R : $R \leftarrow \max_{a \in \Sigma} \underbrace{\text{select}_a(\text{rank}_a(L) + q_a)}_{\text{necessary char's}}$

update L : $L \leftarrow \max_{a \in \Sigma} \underbrace{\text{select}_a(\text{rank}_a(R) - q_a)}_{\text{unnecessary char's}}$ (no match),

$L \leftarrow L + 1$ (match).

Jumping algorithm: update rules

$q = (312)$

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
													L ↓						R ↓		
		b	b	a	c	a	c	c	a	b	a	b	b	a b c c a a				a	c		
a	0	0	1	1	2	2	2	2	3	3	4	4	4	5	5	5	5	6	7	8	8
b	1	2	2	2	2	2	2	2	3	3	4	5	5	6	6	6	6	6	6	6	6
c	0	0	0	1	1	2	3	3	3	3	3	3	3	3	4	5	5	5	5	5	6

update R : $R \leftarrow \max_{a \in \Sigma} \underbrace{\text{select}_a(\text{rank}_a(L) + q_a)}_{\text{necessary char's}}$

update L : $L \leftarrow \max_{a \in \Sigma} \underbrace{\text{select}_a(\text{rank}_a(R) - q_a)}_{\text{unnecessary char's}}$ (no match),

$L \leftarrow L + 1$ (match).

Jumping algorithm: update rules

$q = (312)$

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
	b	b	a	c	a	c	c	a	b	a	b	b	a	b	c	c	a	a	a	c	
a	0	0	1	1	2	2	2	3	3	4	4	4	5	5	5	5	6	7	8	8	
b	1	2	2	2	2	2	2	3	3	4	5	5	6	6	6	6	6	6	6	6	
c	0	0	0	1	1	2	3	3	3	3	3	3	3	3	4	5	5	5	5	6	

update R : $R \leftarrow \max_{a \in \Sigma} \underbrace{\text{select}_a(\text{rank}_a(L) + q_a)}_{\text{necessary char's}}$

update L : $L \leftarrow \max_{a \in \Sigma} \underbrace{\text{select}_a(\text{rank}_a(R) - q_a)}_{\text{unnecessary char's}}$ (no match),

$L \leftarrow L + 1$ (match).

Jumping algorithm: update rules

$q = (312)$

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
														$L \downarrow$						$R \downarrow$	
		b	b	a	c	a	c	c	a	b	a	b	b	a	b	c	c	a	a	a	c
a	0	0	1	1	2	2	2	2	3	3	4	4	4	5	5	5	5	6	7	8	8
b	1	2	2	2	2	2	2	2	3	3	4	5	5	6	6	6	6	6	6	6	6
c	0	0	0	1	1	2	3	3	3	3	3	3	3	3	4	5	5	5	5	5	6

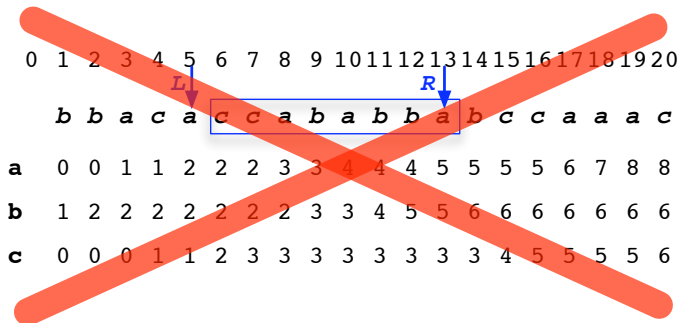
update R : $R \leftarrow \max_{a \in \Sigma} \underbrace{\text{select}_a(\text{rank}_a(L) + q_a)}_{\text{necessary char's}}$

update L : $L \leftarrow \max_{a \in \Sigma} \underbrace{\text{select}_a(\text{rank}_a(R) - q_a)}_{\text{unnecessary char's}}$ (no match),

$L \leftarrow L + 1$ (match).

Jumping algorithm: Analysis

Note that we do not need to store the prefix table, nor the string s .



Jumping algorithm: Analysis

b b a c a c c a b a d d a b c c a a a c



00010110001100110001



110001001000

00011000

Using a wavelet tree [Grossi et al., SODA 2003] as index, we have

- space $O(n)$ (for wavelet tree)
- construction time $O(n \log \sigma)$
- every update (jump) in $O(\sigma)$ time
- query time $O(J\sigma)$, where J = number of jumps (updates)
- **expected** running time: $O(n\sqrt{\frac{\sigma}{\log \sigma} \frac{1}{\sqrt{m}}})$, where $m = \sum_i q_i$.

Jumbled string matching on general alphabets

Latest result: Kociumaka, Radoszewski, Rytter (ESA 2013)

- for arbitrary constant size alphabet
- $o(n^2)$ index size
- $o(n)$ query time (worst-case)
- $O(n^2)$ construction time

(for any $\delta \in (0, 1)$ construct index of size $O(n^{2-\delta})$ with query time $O(m^{\delta(2\sigma-1)})$)

Jumbled string matching on general alphabets

Latest result: Kociumaka, Radoszewski, Rytter (ESA 2013)

- for arbitrary constant size alphabet
- $o(n^2)$ index size
- $o(n)$ query time (worst-case)
- $O(n^2)$ construction time

(for any $\delta \in (0, 1)$ construct index of size $O(n^{2-\delta})$ with query time $O(m^{\delta(2\sigma-1)})$)

Compare to:

1. no index: $O(n)$ space, $O(n)$ query time
2. store all: $O(n^2)$ index size, $O(\log n)$ query time
3. Jumping algo: $O(n)$ index size, $o(n)$ query time **in expectation**

Jumbled string matching on general alphabets

Latest result: Kociumaka, Radoszewski, Rytter (ESA 2013)

- for arbitrary constant size alphabet
- $o(n^2)$ index size
- $o(n)$ query time (worst-case)
- $O(n^2)$ construction time

(for any $\delta \in (0, 1)$ construct index of size $O(n^{2-\delta})$ with query time $O(m^{\delta(2\sigma-1)})$)

Compare to:

1. no index: $O(n)$ space, $O(n)$ query time
2. store all: $O(n^2)$ index size, $O(\log n)$ query time
3. Jumping algo: $O(n)$ index size, $o(n)$ query time **in expectation**

Open problem: Find something better.

4. Prefix normal words

Prefix normal words

Fici, L. (DLT 2011)

Definition

A word $s \in \{a, b\}^*$ is a **prefix normal word** (w.r.t. a) if $\forall 0 \leq m \leq |s|$ no substring of length m has more a 's than the prefix of s of length m .

Example

$s = ababbaabaabbbbaabbab$

$s' = aaababbabaabbababbab$

Prefix normal words

Fici, L. (DLT 2011)

Definition

A word $s \in \{a, b\}^*$ is a **prefix normal word** (w.r.t. a) if $\forall 0 \leq m \leq |s|$ no substring of length m has more a 's than the prefix of s of length m .

Example

$s = ababbaabaabbb**aa**bbab$ *NO*

$s' = aaababbabaabbababbab$ *YES*

Prefix normal forms

Recall $amax_a(m) =$ maximum number of a 's in a substring of s of length m .

$$s = ababbaabaabbbbaabbab$$

m	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
$amax_a(m)$	0	1	2	3	3	4	4	4	5	5	6	7	7	7	8	8	9	9	9	10	10

Theorem

Let $s \in \{a, b\}^*$. Then there exists a unique prefix normal word s' s.t. for all $0 \leq m \leq |s|$, $amax_a(s, m) = amax_a(s', m)$, called its **prefix normal form w.r.t. a** , $PNF_a(s)$.

The **Parikh set** of a word s is the set of Parikh vectors of the substrings of s .

The **Parikh set** of a word s is the set of Parikh vectors of the substrings of s .

Theorem

Two strings $s, t \in \{a, b\}^$ have the same Parikh set if and only if $PNF_a(s) = PNF_a(t)$ and $PNF_b(s) = PNF_b(t)$.*

The **Parikh set** of a word s is the set of Parikh vectors of the substrings of s .

Theorem

Two strings $s, t \in \{a, b\}^*$ have the same Parikh set if and only if $PNF_a(s) = PNF_a(t)$ and $PNF_b(s) = PNF_b(t)$.

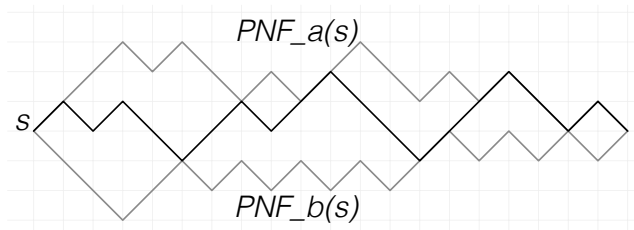


Figure : $/ = a$, $\backslash = b$, $s = ababbaabaabbbaaabbab$ and its prefix normal forms. The region delimited by $PNF_a(s)$ and $PNF_b(s)$ is the **Parikh set** of s .

Computation of PNFs

Open problem: Compute the PNFs in $o(n^2 / \log^2 n)$
 \rightsquigarrow would lead to faster index construction for binary jumbled string matching.

Other variants

- **approximate** jumbled string matching: e.g. find all occurrences between (12, 5, 7) and (8, 2, 4): variant of jumping algorithm expected sublinear (Burcsi, Cicalese, Fici, L., FUN 2010, ToCS 2012)
- (mostly binary) on **trees** and **graphs with bounded treewidth** (Gagie, Hermelin, Landau, Weimann, ESA 2013)
- (mostly binary) locating one occurrence in **strings, trees, graphs** (Cicalese, Gagie, Giaquinta, Laber, L., Rizzi, Tomescu, SPIRE 2013)

THANK YOU!

`zsuzsanna.liptak@univr.it`