

# Computing Matching Statistics and Maximal Exact Matches on Full-Text Indexes

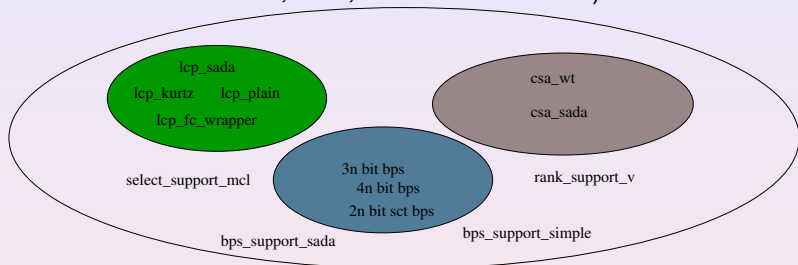
Enno Ohlebusch, Simon Gog, Adrian Kügel

Institute of Theoretical Computer Science  
Ulm University

October 13, 2010

# Motivation

- Myriad of different Compressed Full-Text Indexes exist (different combinations of CSA, LCP, and tree structure)



- Exploit existing Compressed Full-Text Indexes
  - to solve problem with **less memory**
  - and **equal or less time**
 than with **uncompressed** indexes!
- Problem: Find right combination (CSA, LCP, tree structure) for the specific problem.

# Problem 1: Calculate Matching Statistics

## Matching Statistics

Given two strings  $S^1$  and  $S^2$  of length  $n_1$  and  $n_2$ .

A **matching statistics** of  $S^2$  w.r.t.  $S^1$  is an array  $ms$  such that for every entry  $ms[p_2] = (q, [lb..rb])$ ,  $1 \leq p_2 \leq n_2$ , the following holds:

- 1  $\omega = S^2[p_2..p_2 + q - 1]$  is the longest prefix of  $S^2[p_2..n_2 - 1]$  which is substring of  $S^1$ .
- 2  $[lb..rb]$  is the  $\omega$ -interval in the SA of  $S^1$ .

$ms$  was introduced by Chang and Lawler, 1994

## Applications

- String Kernels
- DNA Chips

# Problem 1, Example

$i$	SA	$S^1_{SA[i]}$
0	11	\$
1	3	aaacatat\$
2	4	aacatat\$
3	1	acaaacatat\$
4	5	acatat\$
5	9	at\$
6	7	atat\$
7	2	caaacatat\$
8	6	catat\$
9	10	t\$
10	8	tatat\$
11		

$S^1 = \text{acaaacatat\$}$  and  $S^2 = \text{caaca}$

$ms = (3, [7..7]), (4, [2..2]), (3, [3..4]), (2, [7..8]), (1, [1..6])$

## Problem 2: Calculate MEMs

### Definition of an Exact Match

Given two strings  $S^1$  and  $S^2$  of length  $n_1$  and  $n_2$ .

An **exact match** between  $S^1$  and  $S^2$  is a substring of length  $\ell$  which occurrence starts at position  $p_1$  in  $S^1$  and at position  $p_2$  in  $S^2$ .

Short notation:  $(\ell, p_1, p_2)$ .

### Definition of a Maximum Exact Match (MEM)

An exact match  $(\ell, p_1, p_2)$  is a **maximum exact match** if

- $p_1 = 1$  or  $p_2 = 1$  or  $S^1[p_1 - 1] \neq S^2[p_2 - 1]$  (left maximality)
- $p_1 = n_1$  or  $p_2 = n_2$  or  $S^1[p_1 + \ell] \neq S^2[p_2 + \ell]$  (right maximality)

# Maximum Exact Match, Example

## Example

$S^1 = \text{abraca}dabra$

$S^2 = \text{barricade}$

Not a maximal exact match example:  $a(1, 6, 7)$

# Maximum Exact Match, Example

## Example

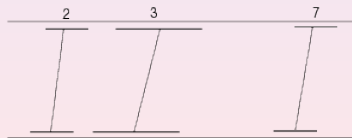
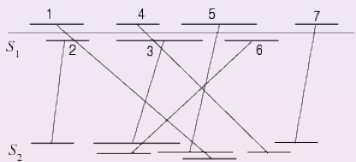
$S^1 = abra\text{cad}abra$

$S^2 = barrica\text{de}$

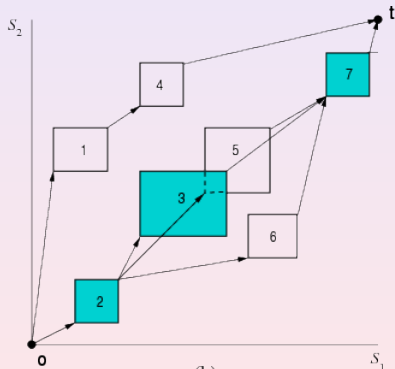
A maximal exact match example: cad (3, 5, 6)

# Applications of MEMs

- sequence analysis
- whole-genome comparisons: e.g. the CoCoNUT software.



(a)



(b)



# Calculating MEMs with a Suffix Tree

## Solution

- Build Suffix Tree of  $S^1 \# S^2 \$$
- Traverse the Suffix Tree in dfs-order
- Search nodes  $v_i$  (depth  $\leq \ell$ ) which subtree contains  $\#$  - and  $\$$  - suffixes and check left maximality

## Drawback

- Space! Best Suffix Tree implementations take about 12-17 bytes per input character.
- **1GB** ASCII text  $\approx$  **12-17GB** Suffix Tree

## Solution

Use compressed index data structures

E.g. Sparse Suffix Arrays (Khan et al. 2009) or Compressed Suffix Trees (this talk)

# Sketch of our MEM solution

- Construct a full-text index for  $S^1$  which provides
  - access to the Burrows and Wheeler Transform (BWT) of  $S^1$
  - access to the longest common prefix (lcp) table
  - parent operation in the CST
- Search all suffixes of  $S^2$  in the full-text index of  $S^1$ 
  - backward search
  - combined with the parent operation

## Result

- Compressed full-text index takes about  $2.375n_1 + \frac{4}{k}n_1$  bytes.
- E.g. 1GB ASCII text and  $k = 16 \approx 2.6\text{GB}$  compressed full-text index

# Component 1: WT + Backward Search

- $backwardSearch(c, [i..j])$  returns interval of pattern or  $\perp$
- $backwardSearch$  step takes  $\mathcal{O}(\log \Sigma)$  time in our implementation
- If  $backwardSearch(c, [i..j]) = \perp$  pattern does not occur in  $S^1$

## Implementation

- BWT is represented by a *wavelet tree* (see Grossi et al.)
- Compressed SA based on wavelet tree and SA samples
- Takes about  $n_1 + \frac{4}{k}n_1$  bytes.

## Component 2: Tree Structure

 $S^1 = \text{a c a a a c a t a t}$ 

$i$	SA	LCP	$S^1_{SA[i]}$	lcp-intervals	
1	3	-1	aaacatat	$0-[1..10]$ $1-[1..6]$ $2-[7..8]$ $1-[9..10]$	$2-[1..2]$
2	4	2	aacatat		$3-[3..4]$
3	1	1	acaaacatat		$2-[5..6]$
4	5	3	acatat		
5	9	1	at		
6	7	2	atat		
7	2	0	caaacatat		
8	6	2	catat		
9	10	0	t		
10	8	1	tat		
11		-1			

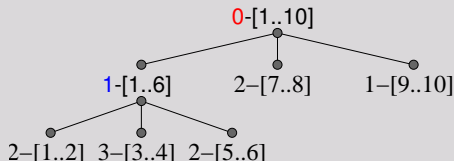
lcp-interval  $\ell - [i..j]$ .

# Component 2: Tree Structure

 $S^1 = \text{acaaacatat}$ 

$i$	$SA$	$LCP$	$S^1_{SA[i]}$
1	3	-1	aaacatat
2	4	2	aacatat
3	1	1	acaaacatat
4	5	3	acatat
5	9	1	at
6	7	2	atat
7	2	0	caaacatat
8	6	2	catat
9	10	0	t
10	8	1	tat
11		-1	

lcp-interval tree



- lcp-interval tree takes  $0.375n_1$  bytes (without lcp-values)
- Parent operation takes constant time

# Component 3: LCP array

- naive solution takes  $n \log n$  bits /  $4n$  bytes
- Sadakane's solution takes  $2n + o(n)$  bits /  $0.26n$  bytes
- pragmatic solution takes 1 byte for small entries and 8 bytes for big entries

## We summarize:

- BWT takes  $n_1$  bytes
- suffix array samples take  $\frac{4}{k}n_1$  bytes
- tree takes  $0.375n_1$  bytes
- lcp values take  $n_1$  bytes

## The new MEM algorithm - example

 $S^1 = \text{acaaacatat}$      $S^2 = \text{caaca}$ 

$i$	SA	BWT	$S^1_{SA[i]}$
1	3	c	aaacatat
2	4	a	aacatat
3	1	t	acaaacatat
4	5	a	acatat
5	9	t	at
6	7	c	atat
7	2	a	caaacatat
8	6	a	catat
9	10	a	t
10	8	a	tat
11			

MEMs of length  $\ell > 1$ 

- $a \Rightarrow (1 - [1..6], p_2 = 5)$
- $ca \Rightarrow 2 - [7..8], p_2 = 4$
- $aca \Rightarrow 3 - [3..4], p_2 = 3$
- $aaca \Rightarrow , p_2 = 2$
- $caa \Rightarrow , p_2 = 1$

Found MEMs ( $\ell, p_1, p_2$ )

backward search

## The new MEM algorithm - example

 $S^1 = \text{acaaacatat}$      $S^2 = \text{caaca}$ 

$i$	SA	BWT	$S^1_{SA[i]}$
1	3	c	aaacatat
2	4	a	aacatat
3	1	t	acaaacatat
4	5	a	acatat
5	9	t	at
6	7	c	atat
7	2	a	caaacatat
8	6	a	catat
9	10	a	t
10	8	a	tat
11			

MEMs of length  $\ell > 1$ 

- $a \Rightarrow (1 - [1..6], p_2 = 5)$
- $ca \Rightarrow 2 - [7..8], p_2 = 4$
- $aca \Rightarrow 3 - [3..4], p_2 = 3$
- $aaca \Rightarrow , p_2 = 2$
- $caa \Rightarrow , p_2 = 1$

Found MEMs ( $\ell, p_1, p_2$ )

backward search



## The new MEM algorithm - example

 $S^1 = \text{acaacatat}$      $S^2 = \text{caaca}$ 

$i$	SA	BWT	$S^1_{SA[i]}$
1	3	c	aaacatat
2	4	a	aacatat
3	1	t	acaacatat
4	5	a	acatat
5	9	t	at
6	7	c	atat
7	2	a	caaacatat
8	6	a	catat
9	10	a	t
10	8	a	tat
11			

MEMs of length  $\ell > 1$ 

- $a \Rightarrow (1 - [1..6], p_2 = 5)$
- $ca \Rightarrow 2 - [7..8], p_2 = 4$
- **$aca \Rightarrow 3 - [3..4], p_2 = 3$**
- $aaca \Rightarrow , p_2 = 2$
- $caa \Rightarrow , p_2 = 1$

Found MEMs ( $\ell, p_1, p_2$ )

backward search

## The new MEM algorithm - example

 $S^1 = \text{acaacatat}$      $S^2 = \text{caaca}$ 

$i$	SA	BWT	$S^1_{SA[i]}$
1	3	c	aaacatat
2	4	a	aacatat
3	1	t	acaacatat
4	5	a	acatat
5	9	t	at
6	7	c	atat
7	2	a	caaacatat
8	6	a	catat
9	10	a	t
10	8	a	tat
11			

MEMs of length  $\ell > 1$ 

- $a \Rightarrow (1 - [1..6], p_2 = 5)$
- $ca \Rightarrow 2 - [7..8], p_2 = 4$
- $aca \Rightarrow 3 - [3..4], p_2 = 3$
- **$aca$**   $\Rightarrow 4 - [2..2], p_2 = 2$
- $caa \Rightarrow , p_2 = 1$

Found MEMs ( $\ell, p_1, p_2$ )

backward search

## The new MEM algorithm - example

 $S^1 = \text{acaacatat}$      $S^2 = \text{caaca}$ 

$i$	SA	BWT	$S^1_{SA[i]}$
1	3	c	aaacatat
2	4	a	aacatat
3	1	t	acaacatat
4	5	a	acatat
5	9	t	at
6	7	c	atat
7	2	a	caaacatat
8	6	a	catat
9	10	a	t
10	8	a	tat
11			

MEMs of length  $\ell > 1$ 

- $a \Rightarrow (1 - [1..6], p_2 = 5)$
- $ca \Rightarrow 2 - [7..8], p_2 = 4$
- $aca \Rightarrow 3 - [3..4], p_2 = 3$
- $aaca \Rightarrow 4 - [2..2], p_2 = 2$
- $caa \Rightarrow , p_2 = 1$
- $caaca \Rightarrow \perp$**

Found MEMs ( $\ell, p_1, p_2$ )

backward search failed

## The new MEM algorithm - example

 $S^1 = acaacatat$      $S^2 = caaca$ 

$i$	SA	BWT	$S^1_{SA[i]}$
1	3	c	aaacatat
2	4	a	aacatat
3	1	t	acaacatat
4	5	a	acatat
5	9	t	at
6	7	c	atat
7	2	a	caaacatat
8	6	a	catat
9	10	a	t
10	8	a	tat
11			

MEMs of length  $\ell > 1$ 

- $a \Rightarrow (1 - [1..6], p_2 = 5)$
- $ca \Rightarrow 2 - [7..8], p_2 = 4$
- $aca \Rightarrow 3 - [3..4], p_2 = 3$
- $aaca \Rightarrow 4 - [2..2], p_2 = 2$
- $caa \Rightarrow , p_2 = 1$
- $BWT[2] = a \neq c = S^2[1]$

Found MEMs  $(\ell, p_1, p_2)$ 

(4, 4, 2)

report MEM

## The new MEM algorithm - example

 $S^1 = \text{acaaacatat}$      $S^2 = \text{caaca}$ 

$i$	SA	BWT	$S^1_{SA[i]}$
1	3	c	aaacatat
2	4	a	aacatat
3	1	t	acaaacatat
4	5	a	acatat
5	9	t	at
6	7	c	atat
7	2	a	caaacatat
8	6	a	catat
9	10	a	t
10	8	a	tat
11			

MEMs of length  $\ell > 1$ 

- $a \Rightarrow (1 - [1..6], p_2 = 5)$
- $ca \Rightarrow 2 - [7..8], p_2 = 4$
- $aca \Rightarrow 3 - [3..4], p_2 = 3$
- $aaca \Rightarrow 2 - [1..2], p_2 = 2$
- $caa \Rightarrow , p_2 = 1$
- $BWT[1] = c = c = S^2[1]$

Found MEMs ( $\ell, p_1, p_2$ )

(4, 4, 2)

Check parent

## The new MEM algorithm - example

 $S^1 = \text{acaacatat}$      $S^2 = \text{caaca}$ 

$i$	SA	BWT	$S^1_{SA[i]}$
1	3	c	aaacatat
2	4	a	aacatat
3	1	t	acaacatat
4	5	a	acatat
5	9	t	at
6	7	c	atat
7	2	a	caaacatat
8	6	a	catat
9	10	a	t
10	8	a	tat
11			

MEMs of length  $\ell > 1$ 

- $a \Rightarrow (1 - [1..6], p_2 = 5)$
- $ca \Rightarrow 2 - [7..8], p_2 = 4$
- $aca \Rightarrow 3 - [3..4], p_2 = 3$
- $aaca \Rightarrow 4 - [2..2], p_2 = 2$
- $caa \Rightarrow , p_2 = 1$
- $SA[3] = 1$

Found MEMs  $(\ell, p_1, p_2)$ 

$(4, 4, 2)$   $(3, 1, 3)$

report MEM

## The new MEM algorithm - example

 $S^1 = \text{acaaacatat}$      $S^2 = \text{caaca}$ 

$i$	SA	BWT	$S^1_{SA[i]}$
1	3	c	aaacatat
2	4	a	aacatat
3	1	t	acaaacatat
4	5	a	acatat
5	9	t	at
6	7	c	atat
7	2	a	caaacatat
8	6	a	catat
9	10	a	t
10	8	a	tat
11			

MEMs of length  $\ell > 1$ 

- $a \Rightarrow (1 - [1..6], p_2 = 5)$
- $ca \Rightarrow 2 - [7..8], p_2 = 4$
- $aca \Rightarrow 3 - [3..4], p_2 = 3$
- $aaca \Rightarrow 4 - [2..2], p_2 = 2$
- $caa \Rightarrow , p_2 = 1$
- $BWT[4] = a = a = S^2[2]$

Found MEMs  $(\ell, p_1, p_2)$ 
 $(4, 4, 2)$   $(3, 1, 3)$ 

 lcp value of parent  $\leq 1$

## The new MEM algorithm - example

 $S^1 = \text{acaacatat}$      $S^2 = \text{caaca}$ 

$i$	SA	BWT	$S^1_{SA[i]}$
1	3	c	aaacatat
2	4	a	aacatat
3	1	t	acaacatat
4	5	a	acatat
5	9	t	at
6	7	c	atat
7	2	a	caaacatat
8	6	a	catat
9	10	a	t
10	8	a	tat
11			

MEMs of length  $\ell > 1$ 

- $a \Rightarrow (1 - [1..6], p_2 = 5)$
- $ca \Rightarrow 2 - [7..8], p_2 = 4$
- $aca \Rightarrow 3 - [3..4], p_2 = 3$
- $aaca \Rightarrow 4 - [2..2], p_2 = 2$
- $caa \Rightarrow , p_2 = 1$
- $BWT[7] = a = a = S^2[3]$

Found MEMs ( $\ell, p_1, p_2$ )

(4, 4, 2) (3, 1, 3)



## The new MEM algorithm - example

 $S^1 = \text{acaacatat}$      $S^2 = \text{caaca}$ 

$i$	SA	BWT	$S^1_{SA[i]}$
1	3	c	aaacatat
2	4	a	aacatat
3	1	t	acaacatat
4	5	a	acatat
5	9	t	at
6	7	c	atat
7	2	a	caaacatat
8	6	a	catat
9	10	a	t
10	8	a	tat
11			

MEMs of length  $\ell > 1$ 

- $a \Rightarrow (1 - [1..6], p_2 = 5)$
- $ca \Rightarrow 2 - [7..8], p_2 = 4$
- $aca \Rightarrow 3 - [3..4], p_2 = 3$
- $aaca \Rightarrow 4 - [2..2], p_2 = 2$
- $caa \Rightarrow , p_2 = 1$
- $BWT[8] = a = a = S^2[3]$

Found MEMs  $(\ell, p_1, p_2)$ 
 $(4, 4, 2)$   $(3, 1, 3)$ 

 lcp value of parent  $\leq 1$

## The new MEM algorithm - example

 $S^1 = \text{acaaacatat}$      $S^2 = \text{caaca}$ 

$i$	SA	BWT	$S^1_{SA[i]}$
1	3	c	aaacatat
2	4	a	aacatat
3	1	t	acaaacatat
4	5	a	acatat
5	9	t	at
6	7	c	atat
7	2	a	caaacatat
8	6	a	catat
9	10	a	t
10	8	a	tat
11			

MEMs of length  $\ell > 1$ 

- $a \Rightarrow (1 - [1..6], p_2 = 5)$
- $ca \Rightarrow 2 - [7..8], p_2 = 4$
- $aca \Rightarrow 3 - [3..4], p_2 = 3$
- $aaca \Rightarrow 4 - [2..2], p_2 = 2$
- $caa \Rightarrow 3 - [7..7], p_2 = 1$

Found MEMs  $(\ell, p_1, p_2)$ 

(4, 4, 2) (3, 1, 3)

continue backward search with  
parent of [2..2] interval

## The new MEM algorithm - example

 $S^1 = acaacatat$      $S^2 = caaca$ 

$i$	SA	BWT	$S^1_{SA[i]}$
1	3	c	aaacatat
2	4	a	aacatat
3	1	t	acaacatat
4	5	a	acatat
5	9	t	at
6	7	c	atat
7	2	a	caaacatat
8	6	a	catat
9	10	a	t
10	8	a	tat
11			

MEMs of length  $\ell > 1$ 

- $a \Rightarrow (1 - [1..6], p_2 = 5)$
- $ca \Rightarrow 2 - [7..8], p_2 = 4$
- $aca \Rightarrow 3 - [3..4], p_2 = 3$
- $aaca \Rightarrow 4 - [2..2], p_2 = 2$
- $caa \Rightarrow 3 - [7..7], p_2 = 1$
- $p_2 = 1$
- $p_2 = 1$

Found MEMs  $(\ell, p_1, p_2)$ 
 $(4, 4, 2)$   $(3, 1, 3)$   $(3, 2, 1)$ 

report MEM

## The new MEM algorithm - example

 $S^1 = \text{acaaacatat}$      $S^2 = \text{caaca}$ 

$i$	SA	BWT	$S^1_{SA[i]}$
1	3	c	aaacatat
2	4	a	aacatat
3	1	t	acaaacatat
4	5	a	acatat
5	9	t	at
6	7	c	atat
7	2	a	caaacatat
8	6	a	catat
9	10	a	t
10	8	a	tat
11			

MEMs of length  $\ell > 1$ 

- $a \Rightarrow (1 - [1..6], p_2 = 5)$
- $ca \Rightarrow 2 - [7..8], p_2 = 4$
- $aca \Rightarrow 3 - [3..4], p_2 = 3$
- $aaca \Rightarrow 4 - [2..2], p_2 = 2$
- $caa \Rightarrow 2 - [7..8], p_2 = 1$

Found MEMs  $(\ell, p_1, p_2)$ 

$(4, 4, 2)$   $(3, 1, 3)$   $(3, 2, 1)$   
 $(3, 6, 1)$

Check parent

## The new MEM algorithm - example

 $S^1 = \text{acaacatat}$      $S^2 = \text{caaca}$ 

$i$	SA	BWT	$S^1_{SA[i]}$
1	3	c	aaacatat
2	4	a	aacatat
3	1	t	acaacatat
4	5	a	acatat
5	9	t	at
6	7	c	atat
7	2	a	caaacatat
8	6	a	catat
9	10	a	t
10	8	a	tat
11			

MEMs of length  $\ell > 1$ 

- $a \Rightarrow (1 - [1..6], p_2 = 5)$
- $ca \Rightarrow 2 - [7..8], p_2 = 4$
- $aca \Rightarrow 3 - [3..4], p_2 = 3$
- $aaca \Rightarrow 4 - [2..2], p_2 = 2$
- $caa \Rightarrow 3 - [7..7], p_2 = 1$

Found MEMs  $(\ell, p_1, p_2)$ 

$(4, 4, 2)$   $(3, 1, 3)$   $(3, 2, 1)$   
 $(3, 6, 1)$

# New MEM algorithm

- Running time:  $\mathcal{O}(n_2 + z \times t_{SA})$ , where  $z$  is the number of right maximal exact matches
- Implementation:
  - Name: backwardMEM
  - Download: [www.uni-ulm.de/in/theo/research/sequana](http://www.uni-ulm.de/in/theo/research/sequana)
  - Experimental comparison vs. sparse SA tool of Khan
  - We measured time and memory for the algorithms not the space for construction

# Our algorithm (backward MEM) vs algorithm of Khan et al. (sparseMEM)

$S^1$	$ S^1 $	$S^2$	$ S^2 $	$\ell$	$K = 4$		$K = 8$	
sparseMEM								
<i>Aspergillus fumigatus</i>	29.8	<i>A.nidulans</i>	30.1	20	4m10s	107	6m44s	74
<i>Homo sapiens21</i>	96.6	<i>Mus m16</i>	35.9	50	12m05s	169	25m01s	163
<i>Mus musculus 16</i>	35.9	<i>Homo s21</i>	96.6	50	6m14s	362	14m15s	255
<i>D. simulans</i>	139.7	<i>D.sechellia</i>	168.9	50	24m20s	490	72m39s	356
<i>D. melanogaster</i>	170.8	<i>D.sechellia</i>	168.9	50	35m02s	588	62m02s	416
<i>D. melanogaster</i>	170.8	<i>D.yakuba</i>	167.8	50	39m21s	586	76m21s	423
backwardMEM					$k = 8$		$k = 16$	
<i>Aspergillus fumigatus</i>	29.8	<i>A.nidulans</i>	30.1	20	58s	89	59s	89
<i>Homo sapiens21</i>	96.6	<i>Mus m16</i>	35.9	50	2m32s	142	2m36s	134
<i>Mus musculus 16</i>	35.9	<i>Homo s21</i>	96.6	50	59s	258	1m15s	225
<i>D. simulans</i>	139.7	<i>D.sechellia</i>	168.9	50	20m11s	399	38m10s	366
<i>D. melanogaster</i>	170.8	<i>D.sechellia</i>	168.9	50	12m50s	504	23m23s	464
<i>D. melanogaster</i>	170.8	<i>D.yakuba</i>	167.8	50	6m08s	510	8m30s	463

Sequence lengths in Mbp, memory in MB

# Experimental results

$S^1$	$S^2$	$\ell$	output size
<i>Aspergillus fumigatus</i>	<i>A.nidulans</i>	20	16MB
<i>Homo sapiens21</i>	<i>Mus musculus 16</i>	50	30MB
<i>Mus musculus 16</i>	<i>Home sapiens21</i>	50	30MB
<i>Drosophila simulans</i>	<i>D.sechellia</i>	50	890MB
<i>Drosophila melanogaster</i>	<i>D.sechellia</i>	50	347MB
<i>Drosophila melanogaster</i>	<i>D.yakuba</i>	50	81MB

Output size (MEMs in mummer-format) of test cases.

## Observation

- running time mainly depends on output size
- bottleneck is **access time to the CSA**



# Problem 1, Solution

```

01  $p_2 \leftarrow n_2 - 1$ 
02  $(q, [i..j]) \leftarrow (0, [0..n_1 - 1])$ 
03 while  $p_2 \geq 0$  do
04    $[lb..rb] \leftarrow \text{backwardSearch}(S^2[p_2], [i..j])$ 
05   if  $[lb..rb] \neq [0..n_1 - 1]$  then
06      $q \leftarrow q + 1$ 
07      $ms[p_2] \leftarrow (q, [lb..rb])$ 
08      $[i..j] \leftarrow [lb..rb]$ 
09      $p_2 \leftarrow p_2 - 1$ 
10   else if  $[lb..rb] = [0..n_1 - 1]$  then
11      $ms[p_2] \leftarrow (0, [1..n_1 - 1])$ 
12      $p_2 \leftarrow p_2 - 1$ 
13   else
14      $q - [i..j] \leftarrow \text{parent}([i..j])$ 

```

# Old vs new approach

## "Traditional" approach

Operations:

- Search forward ( $\mathcal{O}(\log \sigma)$  on (C)ST)
- Suffix Link ( $\mathcal{O}(t_{rmq})$  or  $\mathcal{O}(t_{double\_enclose})$ )

## Backward search approach

Operations:

- search backwards ( $\mathcal{O}(\log \sigma)$  on WT) and map interval to corresponding node in (C)ST: Weiner Link (**time depends on CST**)
- **Parent** on CST ( $\mathcal{O}(1)$  or  $\mathcal{O}(t_{enclose})$ )

# Space for Matching Statistics

## sask-sl

- Fastest program for SK
- Authors: Teo and Vishwanathan
- Uses: SA ( $4n$ ), childtab ( $4n$ ), LCP ( $(1-4)n$ ), Suffix Links ( $8n$ ), text ( $1n$ )

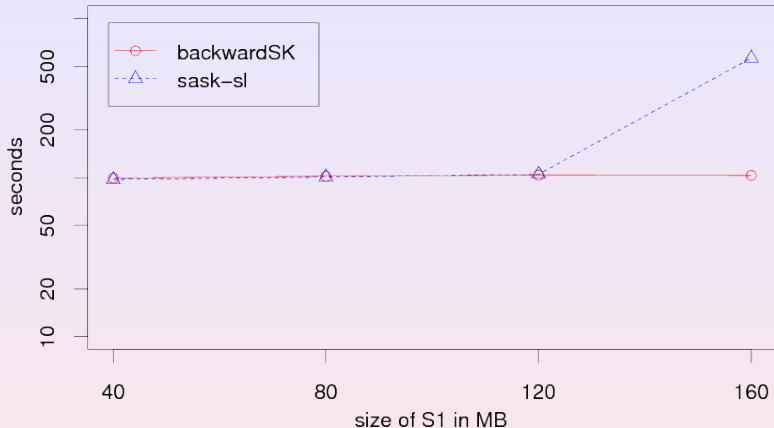
## backwardSK

Uses:

- WT( $1.25n$ )
- LCP ( $(1-4)n$ )
- BPS ( $0.375n$ )

Size $S^1$	backwardSK	sask-sl
40MB	144MB	840MB
80MB	277MB	1680MB
120MB	412MB	2520MB
160MB	539MB	3360MB

# Time for Matching Statistics



## Observation

- running time mainly depends on parent-Operation
- and mapping between lcp-intervals and nodes in the CST

Thank you!  
Any Questions?

# Experimental results

$S^1$	$S^2$	$\ell$	output size
<i>Aspergillus fumigatus</i>	<i>A.nidulans</i>	20	16MB
<i>Homo sapiens21</i>	<i>Mus musculus 16</i>	50	30MB
<i>Mus musculus 16</i>	<i>Home sapiens21</i>	50	30MB
<i>Drosophila simulans</i>	<i>D.sechellia</i>	50	890MB
<i>Drosophila melanogaster</i>	<i>D.sechellia</i>	50	347MB
<i>Drosophila melanogaster</i>	<i>D.yakuba</i>	50	81MB

Output size (MEMs in mummer-format) of test cases.

## Observation

- running time very depends on output size
- bottleneck is access time to the compressed suffix array