Space-efficient genome comparison with compressed full-text indexes

Enno Ohlebusch and Simon Gog

Institut of Theoretical Computer Science
Ulm University

March 24, 2010

Outline

- Problem definition: Finding Maximum Exact Machtes
- Overview: Previous solutions
- Our result
  - Compressed full-text indexes
  - Our new algorithm
- Experimental results
- Conclusion

Definition: Exact Match

Definition of a 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 ℓ which occurence starts at position p_1 in S^1 and at position p_2 in S^2.
Short notation: (ℓ, p_1, p_2) .

Example

$S^1 = abracadabra$
$S^2 = barricade$

Definition: Exact Match

Definition of a 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 ℓ which occurence starts at position p_1 in S^1 and at position p_2 in S^2.
Short notation: (ℓ, p_1, p_2) .

Example

$S^1 = abracadabra$
$S^2 = barricade$
Exact match example: a $(1, 6, 7)$

Definition: Maximum Exact Match (MEM)

Definition of a Maximum Exact Match (MEM)

An exact match (ℓ, 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)

Example

$S^1 = abracadabra$
$S^2 = barricade$
Not a maximal exact match example: a $(1, 6, 7)$

Definition: Maximum Exact Match (MEM)

Definition of a Maximum Exact Match (MEM)

An exact match (ℓ, p_1, p_2) is a maximum exact match if

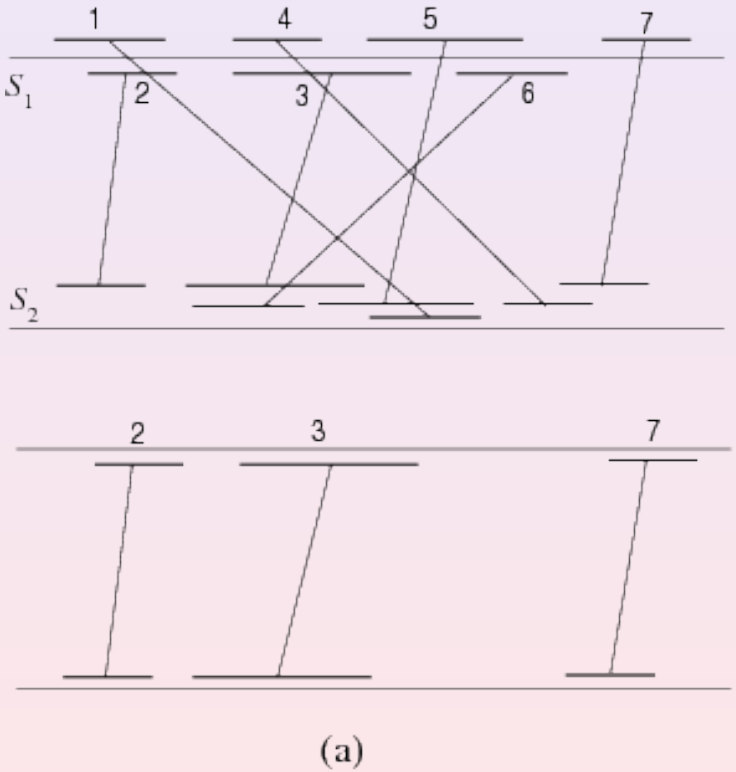
- $p_1 = 1$ or $p_2 = 1$ or $S^1[p_1 - 1] \neq S^2[p_2 - 1]$ (left maximality)
- $p_1 = n_1$ or $p_2 = n_2$ or $S^1[p_1 + \ell] \neq S^2[p_2 + \ell]$ (right maximality)

Example

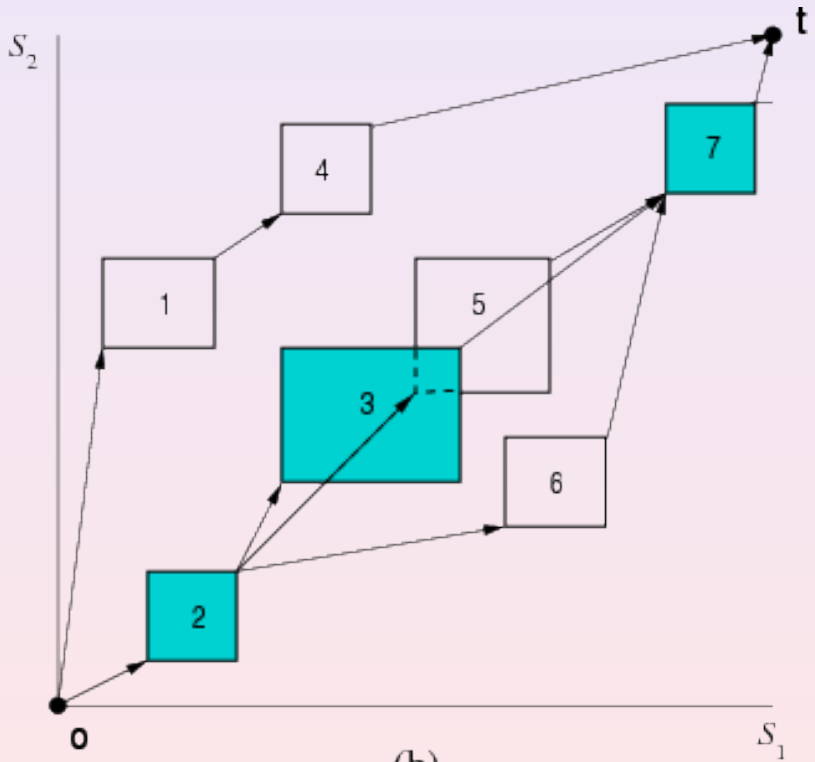
$S^1 = abracadabra$
$S^2 = barricade$
A maximal exact match example: cad $(3, 5, 6)$

Applications of MEMs

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



(a)



(b)

Calculating MEM with a Suffix Tree

Solution

- Build Suffix Tree of $S^1 \# S^2 \$$
- Traverse the Suffix Tree in dfs-order
- Search deepest nodes v_i 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)

Skatch of our solution

- Construct a compressed full-text index for S^1 which provides
  - access to the Burrows-Wheeler-Transformation (BWT) of S^1
  - access to the longest common prefix (lcp) table
  - parent operation in the lcp-interval tree
- Search all suffixes of S^2 in the full-text index of S^1
  - We use 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 $k = 16 \approx$ 2.6GB compressed full-text index

Suffix Array

S^1 = acaaaacatat

i	SA	$S^1_{SA[i]}$
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	tat
11		

Properties

- SA gives the lexicographic order of the suffixes
- pattern matching takes $\mathcal{O}(m \log n)$ (binary search)

Suffix Array + BWT

S^1 = acaaaacatat

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			

Properties

- $BWT[i] = S^1[SA[i] - 1]$
- pattern matching takes $\mathcal{O}(m)$ using backward search

Suffix Array + BWT + Backward Search

S^1 = acaaacatat

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			

Example

- Search pattern acat
- [9..10]
- 4 a's in [1..8], 6 a's in [1..10]
- $[(1+4)..(1+6-1)]$
- 1 c in [1..4], 2 c's in [1..6]
- $[(7+1)..(7+2-1)]$
- 3 a's in [1..7], 4 a's in [1..8]
- $[(1+3)..(1+4-1)]$

Suffix Array + BWT + Backward Search

S^1 = acaaaacatat

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			

Example

- Search pattern acat
- [9..10]
- 4 a's in [1..8], 6 a's in [1..10]
- $[(1+4)..(1+6-1)]$
- 1 c in [1..4], 2 c's in [1..6]
- $[(7+1)..(7+2-1)]$
- 3 a's in [1..7], 4 a's in [1..8]
- $[(1+3)..(1+4-1)]$

Suffix Array + BWT + Backward Search

S^1 = acaaacatat

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			

Example

- Search pattern acat
- [9..10]
- 4 a's in [1..8], 6 a's in [1..10]
- $[(1+4)..(1+6-1)]$
- 1 c in [1..4], 2 c's in [1..6]
- $[(7+1)..(7+2-1)]$
- 3 a's in [1..7], 4 a's in [1..8]
- $[(1+3)..(1+4-1)]$

Suffix Array + BWT + Backward Search

S^1 = acaaacatat

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			

Example

- Search pattern acat
- [9..10]
- 4 a's in [1..8], 6 a's in [1..10]
- $[(1+4)..(1+6-1)]$
- 1 c in [1..4], 2 c's in [1..6]
- $[(7+1)..(7+2-1)]$
- 3 a's in [1..7], 4 a's in [1..8]
- $[(1+3)..(1+4-1)]$

Backward Search

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

Implementation

- BWT is represented with 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.

Enhanced Suffix Array

S^1 = acaaaacatat

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-intervals

- 0-[1..10]
  - 1-[1..6]
    - 2-[1..2]
    - 3-[3..4]
    - 2-[5..6]
  - 2-[7..8]
  - 1-[9..10]

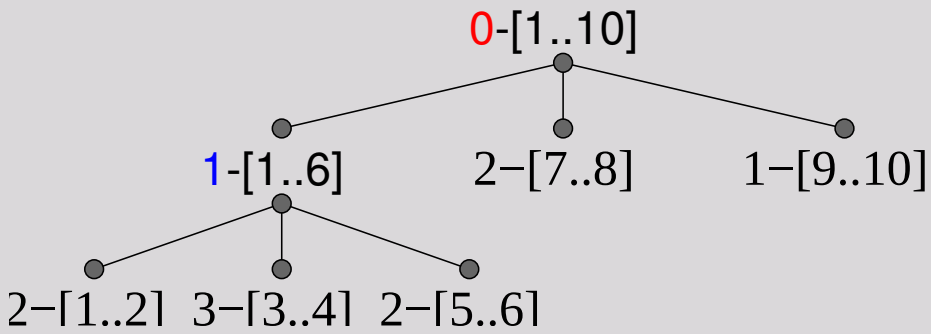
lcp-interval $\ell-[i..j]$. An ℓ-index is an index $k \in [i..j]$ with $LCP[k] = \ell$

Lcp-Interval Tree

S^1 = 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 contstant time

Memory for lcp values

- 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
- lcp-interval tree $0.375 n_1$ bytes
- lcp values take n_1

The new MEM algorithm - example

$S^1 = acaaacatat$ $S^2 = caaca$

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 (ℓ,p_1, p_2)

backward search

The new MEM algorithm - example

$S^1 = acaaacatat$ $S^2 = caaca$

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 (ℓ,p_1, p_2)

backward search

The new MEM algorithm - example

$S^1 = acaaacatat$ $S^2 = caaca$

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 (ℓ,p_1, p_2)

backward search

The new MEM algorithm - example

$S^1 = acaaacatat$ $S^2 = caaca$

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 $\ell > 1$

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

Found MEMs (ℓ,p_1, p_2)

backward search

The new MEM algorithm - example

$S^1 = acaaacatat$ $S^2 = caaca$

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 $\ell > 1$

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

Found MEMs (ℓ,p_1, p_2)

backward search failed

The new MEM algorithm - example

$S^1 = acaaacatat \quad S^2 = caaca$

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 $\ell > 1$

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

Found MEMs (ℓ, p_1, p_2)

$(4, 4, 2)$

report MEM

The new MEM algorithm - example

$S^1 = acaaacatat$ $S^2 = caaca$

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 (ℓ, p_1, p_2)

$(4, 4, 2)$

Check parent

The new MEM algorithm - example

$S^1 = acaaacatat$ $S^2 = caaca$

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 $\ell > 1$

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

Found MEMs (ℓ, p_1, p_2)

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

report MEM

The new MEM algorithm - example

$S^1 = acaaacatat$ $S^2 = caaca$

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 (ℓ, p_1, p_2)

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

lcp value of parent ≤ 1

The new MEM algorithm - example

$S^1 = acaaacatat$ $S^2 = caaca$

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 $\ell > 1$

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

Found MEMs (ℓ,p_1, p_2)

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

The new MEM algorithm - example

$S^1 = acaaacatat$ $S^2 = caaca$

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 $\ell > 1$

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

Found MEMs (ℓ, p_1, p_2)

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

lcp value of parent ≤ 1

The new MEM algorithm - example

$S^1 = acaaacatat$ $S^2 = caaca$

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 (ℓ, p_1, p_2)

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

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

The new MEM algorithm - example

$S^1 = acaaacatat$ $S^2 = caaca$

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 $\ell > 1$

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

Found MEMs (ℓ, p_1, p_2)

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

report MEM

The new MEM algorithm - example

$S^1 = acaaacatat$ $S^2 = caaca$

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 (ℓ, p_1, p_2)

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

Check parent

The new MEM algorithm - example

$S^1 = acaaacatat$ $S^2 = caaca$

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 (ℓ, p_1, p_2)

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

The new MEM algorithm

- Running time: $\mathcal{O}(n_2 + zt_{SA})$, where z is the output size
- Implementation:
  - Name: backwardMEM
  - Download: 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	$\lvert S^1\rvert$	S^2	$\lvert S^2\rvert$	ℓ				
sparseMEM					$K=4$		$K=8$	
Aspergillus fumigatus	29.8	*A.nidulans*	30.1	20	4m10s	107	6m44s	74
Homo sapiens21	96.6	*Mus m16*	35.9	50	12m05s	169	25m01s	163
Mus musculus 16	35.9	*Homo s21*	96.6	50	6m14s	362	14m15s	255
D. simulans	139.7	*D.sechellia*	168.9	50	24m20s	490	72m39s	356
D. melanogaster	170.8	*D.sechellia*	168.9	50	35m02s	588	62m02s	416
D. melanogaster	170.8	*D.yakuba*	167.8	50	39m21s	586	76m21s	423
backwardMEM					$k=8$		$k=16$	
Aspergillus fumigatus	29.8	*A.nidulans*	30.1	20	58s	89	59s	89
Homo sapiens21	96.6	*Mus m16*	35.9	50	2m32s	142	2m36s	134
Mus musculus 16	35.9	*Homo s21*	96.6	50	59s	258	1m15s	225
D. simulans	139.7	*D.sechellia*	168.9	50	20m11s	399	38m10s	366
D. melanogaster	170.8	*D.sechellia*	168.9	50	12m50s	504	23m23s	464
D. melanogaster	170.8	*D.yakuba*	167.8	50	6m08s	510	8m30s	463

Sequence lengths in MBp, Memory in MB

Experimental results

S^1	S^2	ℓ	output size
Aspergillus fumigatus	*A.nidulans*	20	16MB
Homo sapiens21	*Mus musculus 16*	50	30MB
Mus musculus 16	*Home sapiens21*	50	30MB
Drosophila simulans	*D.sechellia*	50	890MB
Drosophila melanogaster	*D.sechellia*	50	347MB
Drosophila melanogaster	*D.yakuba*	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

Any Questions?