Sequence analysis in linear time and compact space

Veli Mäkinen

Helsinki Institute for Information Technology HIIT,
Department of Computer Science, University of Helsinki, Finland

Presenting joint work with Djamal Belazzougui, Fabio Cunial, and Juha Kärkkäinen (ESA 2013) and work by Belazzougui (STOC 2014)

Estonian-Latvian Theory Days
October 3, 2014

MOTIVATION

Sequence analysis is the process of discovering some common features of one or more strings. For example, *maximal repeat* of a string $T = t_1 t_2 \cdots t_n$ is a substring that appears at least twice and whose left and right extensions appear less times.

- `X` is not right-maximal in `agtcXacgatXat` but `Xa` is.

MOTIVATION

Sequence analysis is the process of discovering some common features of one or more strings. For example, *maximal repeat* of a string $T = t_1 t_2 \cdots t_n$ is a substring that appears at least twice and whose left and right extensions appear less times.

- `X` is not right-maximal in `agtcXacgatXat` but `Xa` is.

Maximal unique match (MUM) of two strings A and B is a substring that occurs exactly ones in each string and whose left and right extensions do not appear in both strings.

- `Xa` is a MUM of $A =$ `agtcXa` and $B =$ `cgatXat`.

SOLUTIONS

Suffix tree [Wei73,...] for text of length n from alphabet of size σ:

- $O(n \log n)$ bits
- Myriads of sequence analysis problems in $O(n)$ time

Solutions

Suffix tree [Wei73,...] for text of length n from alphabet of size σ:

- $O(n \log n)$ bits
- Myriads of sequence analysis problems in $O(n)$ time

Compressed suffix tree [Sad07,...]:

- $O(n \log \sigma)$ bits
- Myriads of sequence analysis problems in $O(n \log^{\epsilon} n)$ time

Solutions

Suffix tree [Wei73,...] for text of length n from alphabet of size σ:

- $O(n \log n)$ bits
- Myriads of sequence analysis problems in $O(n)$ time

Compressed suffix tree [Sad07,...]:

- $O(n \log \sigma)$ bits
- Myriads of sequence analysis problems in $O(n \log^{\epsilon} n)$ time

Compressed representations for BWT [GV00,FM00,Sad00,...]

- Kernel of compressed suffix trees
- A few sequence analysis problems in $O(n \log \sigma)$ time

SOLUTIONS

Suffix tree [Wei73,...] for text of length n from alphabet of size σ:

- $O(n \log n)$ bits
- Myriads of sequence analysis problems in $O(n)$ time

Compressed suffix tree [Sad07,...]:

- $O(n \log \sigma)$ bits
- Myriads of sequence analysis problems in $O(n \log^{\epsilon} n)$ time

Compressed representations for BWT [GV00,FM00,Sad00,...]

- Kernel of compressed suffix trees
- A few sequence analysis problems in $O(n \log \sigma)$ time

Compact $O(n \log \sigma)$ bits space and linear time for myriads of problems?

Our ESA 2013 results enhanced with Belazzougui STOC 2014

Compact representations for *bidirectional* BWT:

- $O(n \log \sigma)$ bits
- Many sequence analysis problems in $O(n)$ time

OUR ESA 2013 RESULTS ENHANCED WITH BELAZZOUGUI STOC 2014

Compact representations for *bidirectional* BWT:

- $O(n \log \sigma)$ bits
- Many sequence analysis problems in $O(n)$ time

- Main insights:
  - Conceptual: Visiting suffix tree nodes through suffix link tree $\rightarrow$ No need for LCP array
  - Technical: Avoiding LessThan query on wavelet trees $\rightarrow$ Constant time bidirectional backward step
  - Technical: Index construction in linear time in compact space (Belazzougui, STOC 2014)

Our ESA 2013 results enhanced with Belazzougui STOC 2014

Compact representations for *bidirectional* BWT:

- $O(n \log \sigma)$ bits
- Many sequence analysis problems in $O(n)$ time

- Main insights:
  - Conceptual: Visiting suffix tree nodes through suffix link tree → No need for LCP array
  - Technical: Avoiding LessThan query on wavelet trees → Constant time bidirectional backward step
  - Technical: Index construction in linear time in compact space (Belazzougui, STOC 2014)

Theoretical / practical replacement of compressed suffix trees?

Our results in detail

Representation	1		2		3
Implementation	1a	1b	2a [CPM 2010]	2b	3
Space (bits)	$n \log \sigma +$ $+n + o(n)$	$n \log \sigma +$ $+o(n \log \sigma)$	$2n \log \sigma +$ $+o(n)$	$2n \log \sigma +$ $+o(n \log \sigma)$	$O(n \log \sigma)$
`isLeftMaximal`	$O(\log \sigma)$	$O(1)$	$O(\log \sigma)$	$O(1)$	$O(1)$
`isRightMaximal`	$O(1)$	$O(1)$	$O(\log \sigma)$	$O(1)$	$O(1)$
`enumerateLeft`	$O(\log \sigma)$	$O(1)$	$O(\log \sigma)$	$O(1)$	$O(1)$
`enumerateRight`			$O(\log \sigma)$	$O(1)$	$O(1)$
`extendLeft`	$O(\log \sigma)$	$O(\sigma)$	$O(\log \sigma)$	$O(\sigma)$	$O(1)$
`extendRight`			$O(\log \sigma)$	$O(\sigma)$	$O(1)$
Applications	MUM, SUS, MR, LB, QP, IPS, IPK		MUM, SUS, MEM, SR, NSR, MAW, IPS, IPK		BBB

SUS: shortest unique substrings; MR: maximal repeats; LB: longest border; QP: quasiperiod; IPS: inner product of substrings; IPK: inner product of k-mers; (N)SR: (near) supermaximal repeats; MAW: minimal absent words; BBB: bidirectional b&b (supported also by Implementation 2a).

RELATED WORK

- Bidirectional BWT [Lametal09,SOG10]:
  - Bidirectional backward step in $O(\sigma)$ time [Lametal09] and in $O(\log \sigma)$ time [SOG10].
  - We now improve this to $O(1)$ time (on ranges corresponding to suffix tree nodes).

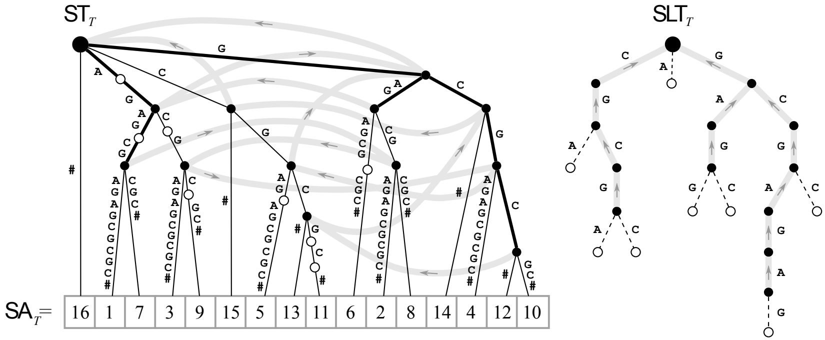
Related work

- Bidirectional BWT [Lametal09,SOG10]:
  - Bidirectional backward step in $O(\sigma)$ time [Lametal09] and in $O(\log \sigma)$ time [SOG10].
  - We now improve this to $O(1)$ time (on ranges corresponding to suffix tree nodes).
- Avoiding LCP array construction to solve *maximal repeats* [BBO12]:
  - Visiting suffix tree nodes in level-wise order.
  - Analysis uses Weiner links.
  - We improve the space and time and show how to solve many related problems.
  - Our technique extends to *synhronized* search and enables indexing for all-against-all problems.

Related work

- Bidirectional BWT [Lametal09,SOG10]:
  - Bidirectional backward step in $O(\sigma)$ time [Lametal09] and in $O(\log \sigma)$ time [SOG10].
  - We now improve this to $O(1)$ time (on ranges corresponding to suffix tree nodes).
- Avoiding LCP array construction to solve *maximal repeats* [BBO12]:
  - Visiting suffix tree nodes in level-wise order.
  - Analysis uses Weiner links.
  - We improve the space and time and show how to solve many related problems.
  - Our technique extends to *synhronized* search and enables indexing for all-against-all problems.
- Alphabet-independent backward search [BN11,BN13]:
  - We extend the technique for bidirectional backward search.

SUFFIX TREE, WEINER LINKS, SUFFIX-LINK TREE



Bidirectional BWT

T=xaby$yabx

T^r=xbay$ybax

sorted suffixes of
T# and T^r#

x	#	x	#
y	$yabx#	y	$ybax#
y	abx#	b	ax#
x	aby$yabx#	b	ay$ybax#
a	bx#	y	bax#
a	by$yabx#	x	bay$ybax#
b	x#	a	x#
#	xaby$yabx#	#	xbay$ybax#
b	y$yabx#	a	y$ybax#
$	yabx#	$	ybax#

character preceding each suffix

#<$<a<b<x<y

BIDIRECTIONAL BWT

T=xaby$yabx

	L	
	x	#
	y	$yabx#
[i,j]	y	abx#
[i,j]	x	aby$yabx#
	a	bx#
	a	by$yabx#
	b	x#
	#	xaby$yabx#
	b	y$yabx#
	$	yabx#

T^r=xbay$ybax

	L’	
	x	#
	y	$ybax#
[i’,j’]	b	ax#
[i’,j’]	b	ay$ybax#
	y	bax#
	x	bay$ybax#
	a	x#
	#	xbay$ybax#
	a	y$ybax#
	$	ybax#

- $i' = i = C[\mathtt{a}]$
- $j' = j = C[\mathtt{a}+1] = C[\mathtt{b}] - 1$
- $L_{i\ldots j} = \mathtt{yx}$
- $L'_{i'\ldots j'} = \mathtt{bb}$

BIDIRECTIONAL BWT

T=xaby\$yabx

x #
y \$yabx#
y abx#
x aby\$yabx#
a bx#
a by\$yabx#
b x#
\# xaby\$yabx#
b y\$yabx#
\$ yabx#

T^r=xbay\$ybax

x #
y \$ybax#
b ax#
b ay\$ybax#
y bax#
x bay\$ybax#
a x#
\# xbay\$ybax#
a y\$ybax#
\$ ybax#

- $i' = i + LessThan_{\mathrm{y}}(L_{i\dots j})$
- $j' = i + LessThan_{\mathrm{y}+1}(L_{i\dots j}) - 1$
- $i = C[\mathrm{y}] + rank_{\mathrm{y}}(L_{1\dots i-1}) + 1$
- $j = C[\mathrm{y}] + rank_{\mathrm{y}}(L_{1\dots j})$

MAXIMAL UNIQUE MATCHES (MUMS)

THEOREM

Substring w is a maximal unique match (MUM) *between $s \in \Sigma^*$ and $t \in \Sigma^*$ iff its only occurrences are $s[i, i + |w| - 1]$ and $t[j, j + |w| - 1]$ and extending w left or right looses one of the occurrences. We can discover all the τ maximal unique matches between s and t in $O(|s| + |t|)$ time and $O((|s| + |t|) \log |\Sigma| + \tau \log(|s| + |t|))$ bits of space.*

- For example, on $s =$ `xaby` and $t =$ `yabx` mums are `x`, `y`, `ab`.

ALGORITHM

Algorithm $\texttt{mums}(M, \texttt{bidirectionalBWTindex}, i, j, i', j', I)$

$\texttt{left} = rank_0(I, j) - rank_0(I, i-1);$

$\texttt{right} = rank_1(I, j) - rank_1(I, i-1);$

if $(\texttt{left} == 0$ **or** $\texttt{right} == 0)$

    return ;

if $(!\texttt{bidirectionalBWTindex.rightMaximal}(i', j'))$

    return ;

if $(\texttt{bidirectionalBWTindex.leftMaximal}(i, j)$ **and** $\texttt{left} == 1$ **and** $\texttt{right} == 1)$

    M is a MUM;

for each $c \in \texttt{bidirectionalBWTindex.EnumerateLeft}(i, j)$ **do**

    $(ii, jj, ii', jj') \leftarrow \texttt{bidirectionalBWTindex.extendLeft}(c, i, j, i', j');$

    $\texttt{mums}(cM, \texttt{bidirectionalBWTindex}, ii, jj, ii', jj', I);$

...

$\texttt{bidirectionalBWTindex}, I \leftarrow \texttt{constructIndex}(s\$t);$

$\texttt{mums}(\texttt{""}, 0, |s| + |t|, 0, |s| + |t|, I);$

ALGORITHM

Algorithm `mums`$(M,$ `bidirectionalBWTindex`$, i, j, i', j', I)$

`left` $= rank_0(I, j) - rank_0(I, i-1)$;
`right` $= rank_1(I, j) - rank_1(I, i-1)$;
if (`left` $== 0$ **or** `right` $== 0$)
  return ;
if (!`bidirectionalBWTindex.rightMaximal`(i', j'))
  return ;
if (`bidirectionalBWTindex.leftMaximal`(i, j) **and** `left` $== 1$ **and** `right` $== 1$)
  M is a MUM;
Recursion with each possible cM...

```
 1234567890
 xaby$yabx
0987654321
         [a]
SA   10 5 7 2 8 3 9 1 4 6
I     1 0 1 0 1 0 1 0 0 1
SA’  10 5 8 3 7 2 9 1 4 6
         [a]
```

ALGORITHM

Algorithm `mums`(M, `bidirectionalBWTindex`, i, j, i', j', I)
(1) `left` $= rank_0(I, j) - rank_0(I, i-1)$;
(2) `right` $= rank_1(I, j) - rank_1(I, i-1)$;
(3) **if** (`left` $== 0$ **or** `right` $== 0$)
(4) **return** ;
(5) **if** (!`bidirectionalBWTindex.rightMaximal`(i', j'))
(6) **return** ;
(7) **if** (`bidirectionalBWTindex.leftMaximal`(i, j) **and** `left` $== 1$ **and** `right` $== 1$)
(8) M is a MUM;
(9) Recursion with each possible cM...

```
 1234567890
 xaby$yabx
0987654321
                [b]
SA   10 5 7 2 8 3 9 1 4 6
I     1 0 1 0 1 0 1 0 0 1
SA'  10 5 8 3 7 2 9 1 4 6
                [b]
```

ALGORITHM

Algorithm `mums`(M, `bidirectionalBWTindex`, i, j, i', j', I)
(1) `left` $= rank_0(I, j) - rank_0(I, i-1)$;
(2) `right` $= rank_1(I, j) - rank_1(I, i-1)$;
(3) **if** (`left` $== 0$ **or** `right` $== 0$)
(4) **return** ;
(5) **if** (!`bidirectionalBWTindex.rightMaximal`(i', j'))
(6) **return** ;
(7) **if** (`bidirectionalBWTindex.leftMaximal`(i, j) **and** `left` $== 1$ **and** `right` $== 1$)
(8) M is a MUM;
(9) Recursion with each possible cM...

```
 1234567890
 xaby$yabx
0987654321
         [ab]
SA   10 5 7 2 8 3 9 1 4 6
I     1 0 1 0 1 0 1 0 0 1
SA'  10 5 8 3 7 2 9 1 4 6
              [ba]
```

ANALYSIS

- Number of recursion steps can be bounded by the amount of explicit and implicit Weiner links in suffix tree, which is linear.
- Claimed space bound follows, except for the use of stack:
  - Must use explicit stack, and push the largest interval first; this guarantees $O(\log n)$ depth.
- Bitvector I can be dropped using *synchronized* bidirectional search on two indexes built on s and t separately.

Analysis

- Number of recursion steps can be bounded by the amount of explicit and implicit Weiner links in suffix tree, which is linear.
- Claimed space bound follows, except for the use of stack:
  - Must use explicit stack, and push the largest interval first; this guarantees $O(\log n)$ depth.
- Bitvector I can be dropped using *synchronized* bidirectional search on two indexes built on s and t separately.

- See the ESA 2013 paper for more involved applications.

Bidirectional step in $O(1)$?

- Bidirectional step requires to count how many symbols smaller than a given symbol there are in a given BWT range (LessThan query).
  - This can be supported by *wavelet tree* in $O(\log \sigma)$ time.
- We show that LessThan query cannot be supported faster than $O(\log \sigma / \log \log n)$ unless using superlinear space.
- However, our algorithms need LessThan query only on ranges corresponding to suffix tree nodes.

Bidirectional step in $O(1)$?

- Bidirectional step requires to count how many symbols smaller than a given symbol there are in a given BWT range (LessThan query).
  - This can be supported by *wavelet tree* in $O(\log \sigma)$ time.
- We show that LessThan query cannot be supported faster than $O(\log \sigma / \log \log n)$ unless using superlinear space.
- However, our algorithms need LessThan query only on ranges corresponding to suffix tree nodes.
- It turns out that $O(1)$ time is possible in this restricted setting.

BIDIRECTIONAL STEP IN $O(1)$

- We extend the technique by Belazzougui and Navarro [BN11,BN13] that supports backward step in constant time for suffix tree node ranges.
- Some ideas:

```
ACGATCGACGAGCTA[CGAGCTAGC]GATCGGCATACGCCGATCGTAC
            C...C
              A...A
           G.....G
             T.......T
```

- There is a representation taking $O(n \log \log \sigma)$ bits that supports partial rank queries in constant time (Belazzougui 2014).
- *Monotone minimal perfect hash function* is required for sorting to derive LessThan answers for maintaining bidirectional BWT range.
- Hashing can be avoided if navigation is *unidirectional*.
- Deterministic linear time construction of BWT and unidirectional BWT index in compact space.
- Most analyses in deterministic linear time.

Discussion

- The techniques also give $O(n)$ time and compact space construction for compressed suffix trees (see Belazzougui, STOC 2014).

DISCUSSION

- The techniques also give $O(n)$ time and compact space construction for compressed suffix trees (see Belazzougui, STOC 2014).
- However, bidirectional or unidirectional BWT index functionality is required to obtain $O(n)$ time sequence analysis on top of such compressed suffix tree.
- There remains a class of sequence analysis tasks that can be solved in $O(n \log^{\epsilon} n)$ time using compressed suffix trees, for which bidirectional BWT index is not sufficient.

Discussion

- The techniques also give $O(n)$ time and compact space construction for compressed suffix trees (see Belazzougui, STOC 2014).

- However, bidirectional or unidirectional BWT index functionality is required to obtain $O(n)$ time sequence analysis on top of such compressed suffix tree.
- There remains a class of sequence analysis tasks that can be solved in $O(n \log^{\epsilon} n)$ time using compressed suffix trees, for which bidirectional BWT index is not sufficient.

- Mäkinen, Belazzougui, Cunial, and Tomescu. *Genome-scale algorithm design: Biological sequence analysis in the era of high-throughout sequencing*. Cambridge University Press. To appear early 2015.