Theoretical aspects of ERa, the fastest practical suffix tree construction algorithm

Matevž Jekovec

University of Ljubljana
Faculty of Computer and Information Science

Oct 10, 2013

Text indexing problem

Text indexing problem

Problem statement

Given unstructured input text T consisting of n characters from alphabet Σ build an index such that for query pattern P we:

- determine whether P occurs in T in time $O(P)$,
- find all occurrences of P in T in time $O(P + occ)$,
- find the longest common prefix (LCP) of P and any suffix of T in time $O(LCP(P, T))$.

Text indexing problem

Problem statement

Given unstructured input text T consisting of n characters from alphabet Σ build an index such that for query pattern P we:

- determine whether P occurs in T in time $O(P)$,
- find all occurrences of P in T in time $O(P + occ)$,
- find the longest common prefix (LCP) of P and any suffix of T in time $O(LCP(P, T))$.

Solution

Suffix tree and *suffix array* (SA) with LCP information are fundamental data structures for indexing unstructured text.

Suffix tree construction algorithms

- Theoretical:

	W ('73), McC ('78)	U ('95)	F-C et al. ('00)
Time	$\Theta(n)$	$\Theta(n)$	$\Theta(n \lg \lvert\Sigma\rvert)$
Online	No	Yes	Yes[1]
Cache-efficient	No	No	Yes
Unbounded Σ	No	No	Yes

[1] Bedathur and Haritsa (2004)

Suffix tree construction algorithms

- Theoretical:

	W ('73), McC ('78)	U ('95)	F-C et al. ('00)
Time	$\Theta(n)$	$\Theta(n)$	$\Theta(n \lg \lvert\Sigma\rvert)$
Online	No	Yes	Yes[1]
Cache-efficient	No	No	Yes
Unbounded Σ	No	No	Yes

- Practical:

	Semi-disk-based			Out-of-core		
	TDD ('04)	ST-Merge ('05)	TRLS. ('07)	B^2ST ('09)	WF ('09)	ERa ('11)
Time	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(n^2)$
Cache-eff.	Yes	Yes	Yes	Yes	Yes	Yes
String acc.	Rnd.	Rnd.	Rnd.	Seq.	Seq.	Seq.
Parallel	No	No	No	No	Yes	Yes

[1] Bedathur and Haritsa (2004)

Suffix tree construction algorithms

- Theoretical:

	W ('73), McC ('78)	U ('95)	F-C et al. ('00)
Time	$\Theta(n)$	$\Theta(n)$	$\Theta(n \lg \lvert\Sigma\rvert)$
Online	No	Yes	Yes[1]
Cache-efficient	No	No	Yes
Unbounded Σ	No	No	Yes

- Practical:

	Semi-disk-based			Out-of-core		
	TDD ('04)	ST-Merge ('05)	TRLS. ('07)	B^2ST ('09)	WF ('09)	ERa ('11)
Time	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(n^2)$
Cache-eff.	Yes	Yes	Yes	Yes	Yes	Yes
String acc.	Rnd.	Rnd.	Rnd.	Seq.	Seq.	Seq.
Parallel	No	No	No	No	Yes	Yes

[1] Bedathur and Haritsa (2004)

Suffix tree construction algorithms

- Theoretical:

	W ('73), McC ('78)	U ('95)	F-C et al. ('00)
Time	$\Theta(n)$	$\Theta(n)$	$\Theta(n \lg \lvert\Sigma\rvert)$
Online	No	Yes	Yes[1]
Cache-efficient	No	No	Yes
Unbounded Σ	No	No	Yes

Huge gap between the theoretical and practical results!

- Practical:

	Semi-disk based			Out-of-core		
	TDD ('04)	ST-Merge ('05)	TRLS. ('07)	B^2ST ('09)	WF ('09)	ERa ('11)
Time	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(n^2)$
Cache-eff.	Yes	Yes	Yes	Yes	Yes	Yes
String acc.	Rnd.	Rnd.	Rnd.	Seq.	Seq.	Seq.
Parallel	No	No	No	No	Yes	Yes

[1] Bedathur and Haritsa (2004)

Suffix tree construction algorithms

- Theoretical:

	W ('73), McC ('78)	U ('95)	F-C et al. ('00)
Time	$\Theta(n)$	$\Theta(n)$	$\Theta(n \lg \|\Sigma\|)$
Online	No	Yes	Yes[1]
Cache-efficient	No	No	Yes
Unbounded Σ	No	No	Yes

Huge gap between the theoretical and practical results!

- Practical:

	Semi-disk based			Out-of-core		
	TDD	ST-Merge	TRLS	B^2ST	WF ('09)	ERa ('11)
Time					$O(n^2)$	$O(n^2)$
Cache-eff.					Yes	Yes
String acc.	Rnd.	Rnd.	Rnd.	Seq.	Seq.	Seq.
Parallel	No	No	No	No	Yes	Yes

Motivation

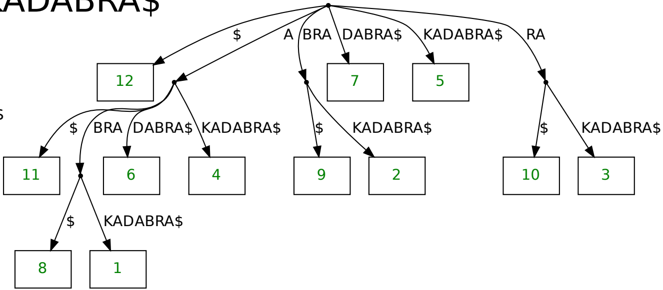
Analyse the fastest practical algorithm and compare it to the theoretical ones.

[1] Bedathur and Haritsa (2004)

Suffix tree

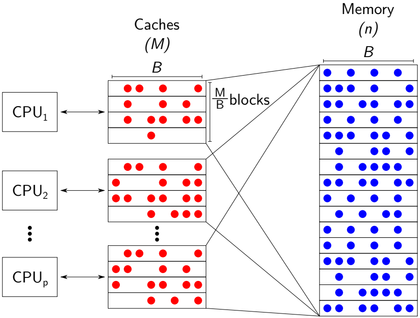
1 2 3 4 5 6 7 8 9 10 11 12
T=ABRAKADABRA$

1 $
2 A$
3 ABRA$
4 ABRAKADABRA$
5 ADABRA$
6 AKADABRA$
7 BRA$
8 BRAKADABRA$
9 DABRA$
10 KADABRA
11 RA$
12 RAKADABRA$



Parallel External Memory model (PEM)[2]

- Shared-memory model describing well the modern multi-core architecture.
- Two-level memory hierarchy w/ fast&limited private caches on 1^{st} level and slow&unlimited main memory on the 2^{nd}.
- Features:
  - P — # of processing elements
  - B — block size
  - M — cache (main memory) size



[2] Arge, Goodrich, Nelson, Sitchinava (2008)

Parallel External Memory model (PEM)

- We assume CREW PEM.
- Data are read and stored on disk and not fitting into main memory:
    - Denote first level as the **main memory** and second level as the **disk**.
- Performance metrics:
    - parallel time,
    - parallel block transfers bw/ disk and main memory.

Suffix tree construction lower bounds

- Sequential:

	bounded Σ	unbounded Σ
Time	$\Omega(Sort(n))$	$\Omega(Sort(n))$
I/Os[3]	$\Omega(Sort(n))$	$\Omega(Sort(n))$
Space[4]	$\Omega(n \lg \lvert\Sigma\rvert)$ bits	$\Omega(n \lg \lvert\Sigma\rvert)$ bits

[3] EM model
[4] Uncompressed index
[5] PEM model

Suffix tree construction lower bounds

- Sequential:

	bounded Σ	unbounded Σ
Time	$\Omega(Sort(n))$	$\Omega(Sort(n))$
I/Os[3]	$\Omega(Sort(n))$	$\Omega(Sort(n))$
Space[4]	$\Omega(n \lg \lvert\Sigma\rvert)$ bits	$\Omega(n \lg \lvert\Sigma\rvert)$ bits

- Parallel on p processing units:

	bounded Σ	unbounded Σ
Parallel time	$\Omega(Sort_p(n))$	$\Omega(Sort_p(n))$
Parallel I/Os[5]	$\Omega(Sort_p(n))$	$\Omega(Sort_p(n))$
Space[4]	$\Omega(n \lg \lvert\Sigma\rvert)$ bits	$\Omega(n \lg \lvert\Sigma\rvert)$ bits

[3] EM model
[4] Uncompressed index
[5] PEM model

Suffix tree construction lower bounds

- Sequential:

	bounded Σ	unbounded Σ
Time	$\Omega(n)$	$\Omega(n \log n)$
I/Os[3]	$\Omega\left(\frac{n}{B}\right)$	$\Omega\left(\frac{n}{B} log_{\frac{M}{B}} \frac{n}{M}\right)$
Space[4]	$\Omega(n \lg \lvert\Sigma\rvert)$ bits	$\Omega(n \lg \lvert\Sigma\rvert)$ bits

- Parallel on p processing units:

	bounded Σ	unbounded Σ
Parallel time	$\Omega\left(\frac{n}{p}\right)$	$\Omega\left(\frac{n}{p} \log n\right)$
Parallel I/Os[5]	$\Omega\left(\frac{n}{pB}\right)$	$\Omega\left(\frac{n}{pB} log_{\frac{M}{B}} \frac{n}{B}\right)$
Space[4]	$\Omega(n \lg \lvert\Sigma\rvert)$ bits	$\Omega(n \lg \lvert\Sigma\rvert)$ bits

[3] EM model
[4] Uncompressed index
[5] PEM model

Suffix tree construction lower bounds

- Sequential:

	bounded Σ	unbounded Σ
Time	$\Omega(n)$	$\Omega(n \log n)$
I/Os[3]	$\Omega\left(\frac{n}{B}\right)$	$\Omega\left(\frac{n}{B} log_{\frac{M}{B}} \frac{n}{M}\right)$
Space[4]	$\Omega(n \lg \lvert\Sigma\rvert)$ bits	$\Omega(n \lg \lvert\Sigma\rvert)$ bits

- Parallel on p processing units:

	bounded Σ	unbounded Σ
Parallel time	$\Omega\left(\frac{n}{p}\right)$	$\Omega\left(\frac{n}{p} \log n\right)$
Parallel I/Os[5]	$\Omega\left(\frac{n}{pB}\right)$	$\Omega\left(\frac{n}{pB} log_{\frac{M}{B}} \frac{n}{B}\right)$
Space[4]	$\Omega(n \lg \lvert\Sigma\rvert)$ bits	$\Omega(n \lg \lvert\Sigma\rvert)$ bits

[3] EM model
[4] Uncompressed index
[5] PEM model

ERa - Elastic Range[7]

- The fastest practical, parallel suffix tree construction algorithm to date.

[6] Heinz, Zobel, Williams (2002)
[7] Mansour, Allam, Skiadopoulos, Kalnis (2011)

ERa - Elastic Range[7]

- The fastest practical, parallel suffix tree construction algorithm to date.
- Time complexity: $O(n^2)$ w.c. — for extremely skewed text!

[6] Heinz, Zobel, Williams (2002)
[7] Mansour, Allam, Skiadopoulos, Kalnis (2011)

ERa - Elastic Range[7]

- The fastest practical, parallel suffix tree construction algorithm to date.
- Time complexity: $O(n^2)$ w.c. — for extremely skewed text!
- Yet, it's **fast** in practice: Constructs and stores the human genome's suffix tree in 14 minutes on 16-core desktop PC w/ HDD storage!

[6] Heinz, Zobel, Williams (2002)
[7] Mansour, Allam, Skiadopoulos, Kalnis (2011)

ERa - Elastic Range[7]

- The fastest practical, parallel suffix tree construction algorithm to date.
- Time complexity: $O(n^2)$ w.c. — for extremely skewed text!
- Yet, it's **fast** in practice: Constructs and stores the human genome's suffix tree in 14 minutes on 16-core desktop PC w/ HDD storage!
- Key idea:
  - DNA, music, proteins are almost random[6]
  - Suppose random input text and PEM model of computation:
    - What is **expected** time complexity?
    - What is **expected** I/O complexity?

---

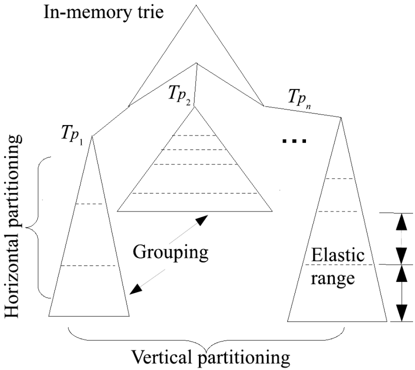
[6] Heinz, Zobel, Williams (2002)

[7] Mansour, Allam, Skiadopoulos, Kalnis (2011)

ERa steps

ERa constructs the suffix tree in two steps:

1. The **vertical partitioning** step determines the suffix subtrees just fitting into M.
2. The **horizontal partitioning** step builds the actual suffix subtrees.



Vertical partitioning

Define **S-prefix** as the prefix of the suffix in the text.

[8] Yue (1991)

Vertical partitioning

Define **S-prefix** as the prefix of the suffix in the text.
Intuition: The S-prefix frequency f_π is # of leaves in the suffix subtree π. This determines the whole subtree size in w.c.!

[8] Yue (1991)

Vertical partitioning

Define **S-prefix** as the prefix of the suffix in the text.
Intuition: The S-prefix frequency f_π is $\#$ of leaves in the suffix subtree π. This determines the whole subtree size in w.c.!
Run sequentially:

1. Scan the text and count the characters frequency $f_\pi : \pi \in \Sigma$.
   - Note: We obtain S-prefixes of length 1.

[8] Yue (1991)

Vertical partitioning

Define **S-prefix** as the prefix of the suffix in the text.
Intuition: The S-prefix frequency f_π is # of leaves in the suffix subtree π. This determines the whole subtree size in w.c.!
Run sequentially:

1. Scan the text and count the characters frequency $f_\pi : \pi \in \Sigma$.
   - Note: We obtain S-prefixes of length 1.
2. For each $\pi : f_\pi > M$, expand character with its right neighbour and count the frequency of obtained S-prefixes (now length 2).

[8] Yue (1991)

Vertical partitioning

Define **S-prefix** as the prefix of the suffix in the text.
Intuition: The S-prefix frequency f_π is # of leaves in the suffix subtree π. This determines the whole subtree size in w.c.!
Run sequentially:

1. Scan the text and count the characters frequency $f_\pi : \pi \in \Sigma$.
   - Note: We obtain S-prefixes of length 1.
2. For each $\pi : f_\pi > M$, expand character with its right neighbour and count the frequency of obtained S-prefixes (now length 2).
3. Repeat step two for S-prefixes of length $3, 4 \dots$ until all f_π fit into the memory M.

[8] Yue (1991)

Vertical partitioning

Define **S-prefix** as the prefix of the suffix in the text.
Intuition: The S-prefix frequency f_π is # of leaves in the suffix subtree π. This determines the whole subtree size in w.c.!
Run sequentially:

1. Scan the text and count the characters frequency $f_\pi : \pi \in \Sigma$.
   - Note: We obtain S-prefixes of length 1.
2. For each $\pi : f_\pi > M$, expand character with its right neighbour and count the frequency of obtained S-prefixes (now length 2).
3. Repeat step two for S-prefixes of length $3, 4 \ldots$ until all f_π fit into the memory M.
4. To optimally fill the main memory combine the S-prefixes into groups fitting into the main memory as tight as possible.
   - Use First-Fit Decreasing heuristic for bin packing problem[8].

[8] Yue (1991)

Vertical partitioning

Analysis:

- For uniform string distributions $O\left(\log_{|\Sigma|} \frac{n}{M}\right)$ scans.
  - Note: The main memory contains $O\left(\frac{n}{M}\right)$ S-prefixes of total size $O(M)$.
- First-Fit Decreasing heuristic requires $O\left(\left(\frac{n}{M}\right)^2\right)$ time.

Vertical partitioning

Analysis:

- For uniform string distributions $O\left(\log_{|\Sigma|} \frac{n}{M}\right)$ scans.
    - Note: The main memory contains $O\left(\frac{n}{M}\right)$ S-prefixes of total size $O(M)$.
- First-Fit Decreasing heuristic requires $O\left(\left(\frac{n}{M}\right)^2\right)$ time.

Overall:

- $O\left(n \log_{|\Sigma|} \frac{n}{M} + \left(\frac{n}{M}\right)^2\right)$ time.
- $O\left(\frac{n}{B} \log_{|\Sigma|} \frac{n}{M}\right)$ I/Os.

Horizontal partitioning

In parallel, for each group of S-prefixes, construct the suffix subtree:

Horizontal partitioning

In parallel, for each group of S-prefixes, construct the suffix subtree:

1. Initialize the optimal length of S-prefixes
   $range = O\left(M/\frac{n}{M}\right) = O\left(\frac{M^2}{n}\right)$
    - Note: The name Elastic Range.

Horizontal partitioning

In parallel, for each group of S-prefixes, construct the suffix subtree:

1. Initialize the optimal length of S-prefixes
   $range = O\left(M/\frac{n}{M}\right) = O\left(\frac{M^2}{n}\right)$
   - Note: The name Elastic Range.
2. Fill buffers with *range* characters following S-prefixes in the text.

Horizontal partitioning

In parallel, for each group of S-prefixes, construct the suffix subtree:

1. Initialize the optimal length of S-prefixes
   $range = O\left(M/\frac{n}{M}\right) = O\left(\frac{M^2}{n}\right)$
   - Note: The name Elastic Range.
2. Fill buffers with *range* characters following S-prefixes in the text.
3. Sort them in lexicographic order and remember their branching information (=LCP) and the original position (=SA).

Horizontal partitioning

In parallel, for each group of S-prefixes, construct the suffix subtree:

1. Initialize the optimal length of S-prefixes
   $range = O\left(M/\frac{n}{M}\right) = O\left(\frac{M^2}{n}\right)$
   - Note: The name Elastic Range.
2. Fill buffers with *range* characters following S-prefixes in the text.
3. Sort them in lexicographic order and remember their branching information (=LCP) and the original position (=SA).
4. While some of the sorted substrings are not unique, repeat steps two and three.
   - Note: Unique strings' buffers are used for non-unique ones in the next iteration, *range* is increasing, the frequency dropping.

Horizontal partitioning

In parallel, for each group of S-prefixes, construct the suffix subtree:

1. Initialize the optimal length of S-prefixes
   $$range = O\left(M/\frac{n}{M}\right) = O\left(\frac{M^2}{n}\right)$$
   - Note: The name Elastic Range.
2. Fill buffers with *range* characters following S-prefixes in the text.
3. Sort them in lexicographic order and remember their branching information (=LCP) and the original position (=SA).
4. While some of the sorted substrings are not unique, repeat steps two and three.
   - Note: Unique strings' buffers are used for non-unique ones in the next iteration, *range* is increasing, the frequency dropping.
5. Construct the suffix subtree using depth-first traversal and reading SA and LCP.

Horizontal partitioning

Analysis:

- Assuming random string distribution, the number of uniquely represented strings in $O(M)$ space is $|\Sigma|^{range}$
  - The number of step two and three iterations is bounded by $O(\log_{|\Sigma|} M)$.
- Each iteration, filling the buffers is done in $O(M)$ time and $O(M/B)$ I/Os.
- Sorting can be done in $O(M)$ time using in-memory radix string sort and requires no I/O.
- Traversing the suffix subtree requires linear $O(M)$ time and $O(M/B)$ I/Os.

Horizontal partitioning

Analysis:

- Assuming random string distribution, the number of uniquely represented strings in $O(M)$ space is $|\Sigma|^{range}$
  - The number of step two and three iterations is bounded by $O(\log_{|\Sigma|} M)$.
- Each iteration, filling the buffers is done in $O(M)$ time and $O(M/B)$ I/Os.
- Sorting can be done in $O(M)$ time using in-memory radix string sort and requires no I/O.
- Traversing the suffix subtree requires linear $O(M)$ time and $O(M/B)$ I/Os.

Overall for constructing $O\left(\frac{n}{M}\right)$ suffix subtrees:

- $O\left(\frac{n}{p}\log_{|\Sigma|} M\right)$ parallel time.
- $O\left(\frac{n}{pB}\log_{|\Sigma|} M\right)$ parallel block transfers.

Conclusion

- ERa despite being practically the fastest algorithm is **not theoretically tight** even for random input strings with uniform substring distribution.

Conclusion

- ERa despite being practically the fastest algorithm is **not theoretically tight** even for random input strings with uniform substring distribution.
- Open problem: Is it possible to design a theoretically tight yet practically competitive algorithm for suffix tree construction?

Conclusion

- ERa despite being practically the fastest algorithm is **not theoretically tight** even for random input strings with uniform substring distribution.
- Open problem: Is it possible to design a theoretically tight yet practically competitive algorithm for suffix tree construction?
- Future work: Analyse ERa bottlenecks in practice and see if they match the critical terms in time and I/O complexities presented here.

Thank you.

Matevž Jekovec

matevz.jekovec@fri.uni-lj.si

http://lusy.fri.uni-lj.si/en/mjekovec

University *of Ljubljana*
Faculty *of Computer and Information Science*

LUSY
Laboratorij za vseprisotne sisteme
Laboratory for Ubiquitous SYstems